Efficient Query Processing in Dynamic Networks of Autonomous Sources

Enrico Franconi[†], Gabriel Kuper[‡], Andrei Lopatenko[†,§]

[†]Free University of Bozen-Bolzano, Faculty of Computer Science, Italy
franconi@inf.unibz.it, lopatenko@inf.unibz.it

[‡]University of Trento, DIT, Italy
kuper@acm.org

[§]University of Manchester, Department of Computer Science, UK

Abstract

In this paper we investigate different optimisation techniques for query processing in networks of autonomous data sources, interconnected in a peer-to-peer fashion by means of GLAV mapping rules at the schema level. There are no restrictions on the topology of the network, in particular, cyclic networks are allowed. We have shown that knowledge of network structure may help to significantly improve the efficiency of query processing, both in the number of exchanged messages and in the time to get a complete answer. However, our optimization methods do not require advance knowledge of this topology, since the topology is discovered during query processing. The contributions of this paper include a definition of soundness and completeness for query processing in dynamically changing networks. Moreover, the query answering algorithm is shown to be efficient with respect to changes of the network. In particular, when the size of the change during query processing is sensibly less than the size of the network itself, query processing time is comparable to that of query processing in a stable network. The assumption that a change is small with respect to the size of whole network is reasonable for large scale networks. The results of an experimental evaluation of the algorithm are presented.

1 Introduction

In this paper, we present a novel efficient distributed query answering algorithm for dynamic networks of autonomous sources organised in a peer to peer structure..

Our proposal shares the spirit of the *Piazza* system [Halevy *et al.*, 2003; Tatarinov and Halevy, 2004]. The vision of the Piazza peer data management system (PDMS) project is to provide semantic mediation between an environment of autonomous and heterogeneous peers, each with its own schema. Rather than requiring the use of a single, uniform, centralised mediated schema to share data between peers, Piazza allows peers to define schema mappings between pairs of peers (or among small subsets of peers). In turn, transitive relationships among the schemas of the peers are exploited so the entire resources of the PDMS can be used. The original Piazza system is limited in the fact that it does not allow complex mapping rules (i.e., schema mappings must be safe rules with atomic heads), it does not allow for fully cyclic mapping rules, and it does not allow for dynamic networks (i.e., networks where peers may join or leave anytime).

The problem of a PDMS with autonomous and heterogeneous peers exemplified by Piazza is different from the *structured P2P systems*, such as [Stoica *et al.*, 2001; Ratnasamy *et al.*, 2001]. The peers in structured P2P systems form an overlay network that has some superimposed logical structure, such as a ring in Chord, a d-dimensional coordinate space in CAN or a Skip Graph data structure. This logical overlay structure among the peers can be utilised to efficiently route a query. This is achieved by means of distributed hash tables mapping data and queries to the logical structure of the network. For example, in Chord and Skip Graphs the queries can be routed in $O(\log n)$ overlay hops in a network of n peers. This can be achieved only if the schemas of the peers are (almost) homogeneous and known in advance, so that the distributed hash tables can be effectively computed. This is not the case of the scenario of a PDMS with autonomous and heterogeneous peers, where peers may join the network by autonomously deciding how to map their own heterogeneous schema with other schemas of arbitrarily chosen acquaintances, each one possibly with a different schema. As a consequence, it is evident that in each communication step between two peers the data is transformed according to the specific semantic schema mapping relating them; therefore, when some remote data is required to answer a query, the data has to flow through all the intermediate peers in order to be interpreted correctly by appropriate transformations.

Together with the work presented in [Halevy *et al.*, 2003; Tatarinov and Halevy, 2004], other researchers investigated the theoretical underpinnings of peer database management systems. The work presented in [Calvanese *et al.*, 2004] proposes a logical analysis of the theory behind a PDMS, but it lacks a distributed algorithm: it assumes that nodes may exchange both data *and* mappings, so that only the query node will eventually evaluate the query answer in one go – there is no distributed computation and the network may be flooded with data. The work presented in [Bernstein *et al.*, 2002; Serafini *et al.*, 2003] proposes a very general theoretical framework for PDMS, with expressive schema mapping languages (up to first order logic) and constraint languages (up to first order logic) applied to single peers. However, no computational characterisation is given. The paper [Serafini and Ghidini, 2000] describes a local algorithm to compute query answers in a P2P network, but it allows only safe schema mapping rules with atomic heads. The algorithm is exponential in the number of nodes and it floods the network with messages during query evaluation if the network contains cycles. None of the above PDMS approaches supports dynamic networks: in the case of peers joining or leaving the network during the computation, neither the termination of the query answering algorithm nor the properties of the possible query answer are guaranteed.

Starting from the general ideas sketched above, the paper [Franconi *et al.*, 2003] introduces a general logical and computational characterisation of networks of autonomous sources, interconnected by means of schema mapping rules between pairs of peers. This paper defines a precise model-theoretic semantics of a PDMS (fully compatible with Piazza and the other PDMS framework presented above), it characterises the general computational properties for the problem of answering queries to a PDMS, and it presents tight complexity bounds and basic distributed procedures for important special cases. The paper [Franconi *et al.*, 2004a] analyses a distributed procedure for the problem of local database update in a network of database peers. The problem of local database update is different from the problem of query answering. Given a PDMS, the answer to a local query may involve data that is distributed over the network, and this may require the participation of many nodes at query time. On the other hand, given a PDMS, a "batch" update algorithm will be such that all the nodes consistently and optimally propagate all the relevant data to their neighbours, allowing for subsequent local queries to be answered locally within a node, without fetching data from other nodes at query time. The update problem has been considered important by the P2P literature; most notably, recent papers focused on the importance of data exchange and materialisation for a stable P2P network [Fagin *et al.*, 2003; Daswani *et al.*, 2003]. The papers [Franconi *et al.*, 2004b] introduce a basic distributed algorithm for query answering in a PDMS, together with the coDB prototypical implementation in the JXTA framework. The proposed algorithm is polynomial in data complexity, but it is still exponential in the dimension of the network. These papers consider a network of databases, possibly with different schemas, interconnected by means of mapping rules having conjunctive queries both in the body and in the head, with possibly existential variables both in the body and in the head (called GLAV rules) as first suggested by [Calvanese *et al.*, 2004]. Each node can be queried with a conjunctive query over its schema, for data which the node can possibly fetch from its neighbours using appropriate mapping rules. Unrestricted cyclic topologies of the network are allowed. The proposed PDMS framework is robust in the sense that it supports *dynamic* networks: even if nodes and mapping rules appear or disappear during the computation, the proposed algorithm will eventually terminate with a provably sound and complete result.

Some work on distributed data replication uses similar techniques, such as "lazy replication" and "epidemic algorithms" [Holliday *et al.*, 2003], but these are not directly applicable to the kind of PDMS considered by Piazza and by the other references presented above, since they rely on a different semantics of the mappings. Similarly, the *routing indexes* technology presented by [Crespo and Garcia-Molina, 2002] can not be applied to the kind of PDMS with cyclic mappings and dynamic networks, since the presence of cycles in the network together with the a-priori ignorance of the (dynamic) network topology invalidates any careful selection of neighbours providing answers. On the positive side, we are considering to integrate into our PDMS framework the idea of *data mappings* as described in [Kementsietsidis *et al.*, 2003; Kementsietsidis and Marcelo Arenas, 2003], which introduces a table that maintains mappings to the neighbour's data, to mimic a sort of extensional constraint.

1.1 PDMS vs. Data Integration Systems

Another line of research that is necessary to compare with the PDMS framework proposed here, is the standard classical logic-based data integration technology, which has been summarised in a very clear way by [Lenzerini, 2002]; successful examples of classical logic-based data integration technology are the Information Manifold [Kirk *et al.*, 1995] and Tsimmis [Garcia-Molina *et al.*, 1997]. The main difference is in the role of the schema mapping rules between nodes: in a PDMS a schema mapping rule is intended for data migration and transformation between neighbours, as opposed to the role of global logical constraints in classical data integration systems. It can be proved (see, e.g., [Franconi *et al.*, 2003]) that by adopting a PDMS semantics the complexity of query answering is reduced from exponential (or undecidable) down to polynomial.

Let's explain by means of an example why the PDMS semantics is different from the classical semantics given to data integration systems. Suppose we have three distributed databases. The first one (DB_1) is the municipality's internal database, which has a binary table `Citizen-1` which contains the name of the citizen and the marital status (with values *single* or *married*). The sec-

ond one (DB_2) is a public database, obtained from the municipality's database, with two unary tables Male-2 and Female-2. The third database (DB_3) is the Pension Agency database, obtained from a public database, with the unary table Citizen-3 and a binary table Marriage-3 (stating that two people are married). The three databases are interconnected by means of the following rules:

$1:$ Citizen-1$(x, y) \Rightarrow 2:$ (Male-2$(x) \vee$ Female-2(x))
(this rule connects DB_1 with DB_2)

$2:$ Male-2$(x) \Rightarrow 3:$ Citizen-3(x)
$2:$ Female-2$(x) \Rightarrow 3:$ Citizen-3(x)
(these rules connect DB_2 with DB_3)

In the classical logical model, the Citizen-3 table in DB_3 should be filled with all of the individuals in the Citizen-1 table in DB_1, since the following rule is logically implied:

$1:$ Citizen-1$(x) \Rightarrow 3:$ Citizen-3(x)

However, in a system of autonomous sources this is not a desirable conclusion. In fact, rules should be interpreted only for fetching data, and not for logical computation. In this example, the tables Female-2 and Male-2 in DB_2 will be empty, since the data is fetched from DB_1, where the gender of any specific entry in Citizen-1 is not known. From the perspective of DB_2, the only thing that is known is that each citizen is in the view (Female-2 $\vee$ Male-2). Therefore, when DB_3 asks for data from DB_2, the result will be empty.
In other words, the rules

$2:$ Male-2$(x) \Rightarrow 3:$ Citizen-3(x)
$2:$ Female-2$(x) \Rightarrow 3:$ Citizen-3(x)

will transfer no data from DB_2 to DB_3, since no individual is known in DB_2 to be either definitely a male (in which case the first rule would apply) or definitely a female (in which case the second rule would apply). We only know that any citizen in DB_1 is either male or female in DB_2, and no reasoning about the rules should be allowed.

In order to explain the importance of cyclic rules, suppose now to have an additional cyclic pair of rules connecting DB_1 and DB_3 as follows:

$1:$ Citizen-1$(x,$ "married"$) \Rightarrow 3:$ Marriage-3(x, y)
$3:$ Marriage-3$(x, y) \Rightarrow$
$1:$ Citizen-1$(x,$ "married"$) \wedge$
$1:$ Citizen-1$(y,$ "married"$)$

These rules serve the purpose to synchronise the people who are known to be married in DB_1 (by means of the Citizen-1 table) with the people who are known to be married in DB_3 (by means of the Marriage-3 table). Suppose that it is known in DB_1 that only John is married, and nothing is known in DB_3 about marriages. In the classical logical model, a query to DB_3 asking for the non existence of some married person different from John will get a negative answer[1]. In a PDMS setting, we actually expect a positive answer, since the only information that is fetched is about John.

[1]Note that the semantics of a query is the *certain answer* semantics.

1.2 Our Contribution

The main contribution of this paper is to extend the results presented in [Franconi *et al.*, 2003; 2004a; 2004b], by introducing and evaluating experimentally a fully distributed query processing algorithm for a PDMS, which is *polynomial* both in data complexity and in the dimension of the network. As it comes out from the comparison with the unoptimized version of the algorithm, the new version of the algorithm outperforms the unoptimized one exponentially with respect to the size of the network both in message and time complexity for highly connected networks. The new proposed algorithm handles dynamic networks, and it works without any starting assumption about the topology of the network.

The paper is organised as follows: first, we introduce some general definitions of a PDMS, and of its dynamic behaviour. Then we describe the distributed query answering algorithms, by emphasising the optimisations that lead to the polynomial complexity. At the end, we present an experimental evaluation of the algorithm on some real and random cases.

2 PDMS network of Heterogeneous Databases

We quickly define here a PDMS (also called here a network of autonomous sources), which is basically a collection of local databases together with schema mapping rules that interconnect databases pairwise.

Definition 1 (Local database) *Let I be a nonempty finite set of indexes $\{1, 2, \ldots, n\}$, and C be a set of constants. For each pair of distinct i, $j \in I$, let L_i be a first-order logic without function symbols, with signature disjoint from L_j but for the shared constants C. A* local database *DB_i is a theory on the first order language L_i.*

Nodes are interconnected by means of mapping rules. A mapping rule allows a node i to fetch data from its neighbour nodes $j_1, \ldots, j_m$.

Definition 2 (Schema mapping rule) *A* mapping rule *is an expression of the form*

$$j_1 : b_1(\mathbf{x}_1, \mathbf{y}_1) \wedge \cdots \wedge j_k : b_k(\mathbf{x}_k, \mathbf{y}_k) \Rightarrow i : h(\mathbf{x}, \mathbf{y})$$

where $j_1, \ldots, j_k$, i are distinct indices, each $b_l(\mathbf{x}_l, \mathbf{y}_l)$ is a formula of L_{j_l}, and $h(\mathbf{x}, \mathbf{y})$ is a formula of L_i, and $\mathbf{x} = \mathbf{x}_1 \cup \cdots \cup \mathbf{x}_k$.

Note that we are making the simplifying assumption that equal constants mentioned in different nodes refer to the same objects, i.e., that they play the role of URIs (Uniform Resource Identifiers); this is the underlying approach of the Semantic Web framework, for example. Other approaches consider *domain relations* to map objects between different nodes [Serafini *et al.*, 2003], and we plan to consider such extensions to our model in the future. Note, that predicate

names are always distinct in distinct peers; this can be seen as modelling the notion of namespaces.

A PDMS system is the collection of nodes interconnected by (possibly cyclic) rules.

Definition 3 (PDMS system) *A* Peer Database Management System (PDMS) *is a tuple of the form* $\mathcal{MDB} = \langle \mathcal{I}, \mathcal{C} \rangle$*, where* $\mathcal{I} = \{DB_1, \cdots, DB_n\}$ *is the set of local databases, and* $\mathcal{C}$ *is the set of mapping rules.*

A user accesses the information hold by a PDMS by formulating a query at a specific node.

Definition 4 (Query) *A* local query *is a first order formula in the language of one of the local databases* DB_i*.*

In this paper we restrict the general framework as follows:

- all the nodes are plain relational databases without constraints;
- schema mapping rules may contain conjunctive queries in both the head and body (without any safety assumptions and possibly with built-in predicates in the body);
- the body involves only one node per rule;
- queries are just conjunctive queries.

In this paper we don't give the details of the logic based formalism of the PDMS and of queries, as it has been thoroughly analysed in, e.g., [Franconi *et al.*, 2003; Calvanese *et al.*, 2004].

To describe the query processing in a PDMS we need to introduce the notion of a *dependency edge* between peers of a network of autonomous sources.

Definition 5 *There is a* dependency edge *from a peer i to peer j, if there is a mapping rule with head at peer i and body at peer j.*

Note that the direction of a dependency edges is the opposite of that of the rules. The direction of a rule is the direction in which data is transfered, whereas the dependency edge has the opposite orientation. In this paper we use $\mathcal{MDB}$ to denote a PDMS, using terms such as *PDMS* or *a network*; please note that we consider the general case when the network is *cyclic*. $\mathcal{I}$ is used to denote a set of all peers in given $\mathcal{MDB}$, $\mathcal{C}$ denotes the set of all mappings, and $\mathcal{L}$ the set of dependency edges between peers in a network derived from $\mathcal{C}$. Subsets of $\mathcal{I}$ are denoted by $\mathcal{A}$. We assume that $\mathcal{I}$, $\mathcal{L}$, and $\mathcal{C}$ are always finite sets.

Definition 6 *A dependency path for a peer i is a path* $\langle i_1, i_2, \ldots, i_n \rangle$ *of dependency edges, such that 1)* $i_1 = i$*; 2)* $\langle i_1, \ldots, i_{n-1} \rangle$ *is a simple path (no one peer appears twice).*

Definition 7 *A maximal dependency path for a peer i is a dependency path such that it is impossible to add a new peer to the path or if we add any peer to the path, the result will not be a closed dependency path. In this paper, when we describe dependency paths for a peer i, we omit the first peer (i).*

There are three types of maximal dependency paths starting in *i*:

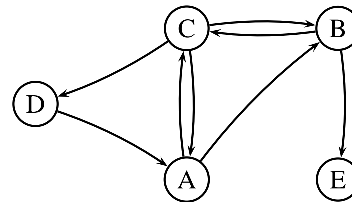
1. a directed acyclic walk, which starts in *i* and ends in a leaf peer of the network. Example: a path ABE in a pic.2;
2. a semi-cyclic walk which consist of a directed acyclic walk which starts in *i* and ends in a some peer w and a directed cycle which starts in w and ends in w. Example: a path $ABCB$, which consist of an acyclic path AB and a cycle BCB;
3. a directed cycle which starts and ends in *i*. Example: a path $ABCA$.

As an example, consider a PDMS with the following schemas and rules:

$$\begin{array}{rcl} A & : & a(X,Y) \\ B & : & b(X,Y) \\ C & : & c(X,Y), f(X) \\ D & : & d(X,Y) \\ E & : & e(X,Y) \end{array}$$

$$\begin{array}{rcl} r1 & : & E : e(X,Y) \rightarrow B : b(X,Y) \\ r2 & : & B : b(X,Y), b(Y,Z) \rightarrow C : c(X,Z) \\ r3 & : & C : c(X,Y), c(Y,Z) \rightarrow B : b(X,Z) \\ r4 & : & B : b(X,Y), b(X,Z), X \neq Z \rightarrow A : a(X,Y) \\ r5 & : & A : a(X,Y) \rightarrow C : f(X) \\ r6 & : & A : a(X,Y) \rightarrow D : d(Y,X) \\ r7 & : & D : D(X,Y), D(Y,Z) \rightarrow C : c(X,Y) \\ r8 & : & C : f(X) \rightarrow A : A(X,X) \end{array}$$

The dependency edges and the maximal dependency paths for the example above are:



#	path	#	path	#	path	#	path
A	ABCA	B	BE	C	BE	D	ABE
A	ABE	B	BCAB	C	BC	D	ABCD
A	ABCB	B	BCB	C	DABC	D	ABCB
A	ABCA	B	BCDAB	C	ABC	D	ABCA
				C	ABE		

3 Dynamic behaviour of the network of autonomous sources

One of the distinctive characteristics of PDMS systems is that the network can vary dynamically. Assume that the network $\mathcal{MDB}$ consist initially of a set of nodes $\mathcal{I}$, and that $\mathcal{C}$ is an initial set of mappings with $\mathcal{L}$ being the initial set of dependency edges. We model network dynamicity by adding/removing the mappings between nodes; deletion of a node is therefore modelled by deleting all mapping rules that relate to this node. With respect to query answering adding/removing nodes with mappings is easily seen to be equivalent to the assumption that all nodes are present from the beginning, and that only the mappings may change.

We define an atomic network change operation as follows.

- *addLink(i,j,rule,id)*: add the mapping rule *rule* from node *j* (the body) to node *i* (the head). *id* is the name of a rule, which should be unique for a given pair of nodes.
- *deleteLink(i,j,id)*: delete the mapping rule *id* between nodes *i* and *j*

Definition 8

1. *A* change $\mathbf{U}$ of a network $\mathcal{MDB}$ *is a sequence of atomic change operations over* $\mathcal{MDB}$.
2. *A* finite change *of a network is a finite sequence of atomic changes.*
3. *An* initial subchange $\mathbf{U}_1$ *of a change* $\mathbf{U}$ *is an initial prefix of* $\mathbf{U}$.
4. *A* subchange $\mathbf{U}_{\mathcal{A}}$ *of* $\mathbf{U}$ *in respect to* $\mathcal{A} \subset \mathcal{I}$ *is a set of atomic operations of* $\mathbf{U}$*, relevant to* $\mathcal{A}$ *and ordered with the same order as in* $\mathbf{U}$.

We assume that in the case of an atomic change the network will be notified about the change in the following way:

1. in case of *addLink(i,j,rule,id)*, the node *i* (which will fetch data through this rule) gets the notification *addRule(i, j, rule, id)*
2. in case of *deleteLink(i,j,id)*, the node *i* (which will be unable to fetch data through this rule) gets the notification *deleteRule(i, j, id)*

Assume that $\preceq$ is a partial ordering relation on the set of dependency edges of the PDMS network: $E_1 \preceq E_2$ iff there is a directed path from the head of E_2 to the tail of E_1.

Definition 9

1. *A change* $\mathbf{U}_1 \preceq_{complete} \mathbf{U}_2$ *(*$\mathbf{U}_1$ *is less wrt the completeness relation than* $\mathbf{U}_2$*), iff there is no* $deleteLink(i, _) \in \mathbf{U}_1$*, such that there exist* $deleteLink(j, _) \in \mathbf{U}_2$*, and* $j \preceq i$ *and* $deleteLink(j, _)$ *does not occur after* $deleteLink(i, _)$.
2. *A change* $\mathbf{U}_1 \preceq_{sound} \mathbf{U}_2$ *(*$\mathbf{U}_1$ *is less wrt the soundness relation than* $\mathbf{U}_2$*), iff there is no* $addLink(i, _) \in \mathbf{U}_1$*, such that there exist* $addLink(j, _) \in \mathbf{U}_2$*, and* $j \preceq i$ *and* $deleteLink(j, _)$ *does not occur after* $deleteLink(i, _)$.

Definition 10

1. *A* sound answer *of a query Q in a network subject to runtime changes with respect to a network change* $\mathbf{U}$*, is an answer to the query that is included in the result that we would obtain if we executed all the* $addLink$ *statements before running Q, and did not execute the* $deleteLink$ *statements at all.*
2. *A* complete answer *of a query Q in a network subject to runtime changes, is an answer to the query that contains the result that we would obtain if we executed all the* $deleteLink$ *statements before running Q, and did not execute the* $addLink$ *statements at all.*

The motivation behind this definition is that we cannot know in advance what the state of the database will be at termination, since the changes may happen at any moment during the query answering algorithm execution. Therefore, we require that a sound and/or complete answer will be classically sound and/or complete with respect to the part of the network that is *unchanged*. The result with respect to the part that is changed will depend on the order and timing of the execution of the changes. In this sense, the answer to a query in a network subject to "small" changes will be still meaningful with respect to the majority of the data that resides in the stable parts of the network.

In general, we expect that a query answering algorithm in a PDMS dynamic network enjoys the following properties, which will be satisfied by the algorithm we propose in this paper.

Theorem 1

1. *(Soundness and completeness) For a finite change of a network, the query algorithm terminates with a sound and complete answer.*
2. *(Termination) In the case of an infinite change to the network, the query algorithm may not terminate.*
3. *(Complexity) For a finite runtime change of the network, the complexity of the query algorithm at each node is at most quadratic with respect to the size of the change.*

In many cases, we will not be able to assume that a network change is finite. In the general case, therefore, the nodes in the network may never reach the fix-point – or at least, we may not be able to show that they have reached a fix-point.

Definition 11

1. *A set of nodes $\mathcal{A}_1$ is separated from a set of nodes $\mathcal{A}_2$ in a network of autonomous sources if there is no dependency path from any node in $\mathcal{A}_1$ that involves a node in $\mathcal{A}_2$.*

2. *A set of nodes $\mathcal{A}_1$ is separated from a set of nodes $\mathcal{A}_2$ in PDMS network with respect to a change* **U** *if for any subchange of* **U** *there is no dependency path from a node in $\mathcal{A}_1$ involving a node in $\mathcal{A}_2$ in the network we obtain by applying that subchange.*

4 The Query Processing Algorithm

We now describe our algorithm for distributed query processing in a a network of autonomous sources. The algorithm is based on asynchronous messaging. The asynchronous model assumes a guaranteed delivery of each message, and the preservation of the order of the messages during inter-peer communications. No assumption is required about the ordering of message delivery globally. The algorithm is invoked by issuing a query to some node, and it terminates when the node provides the complete answer to the query.

In comparison with other algorithms handling cyclic networks presented in the literature (e.g., [Franconi *et al.*, 2004a; 2004b; Serafini and Ghidini, 2000; Calvanese *et al.*, 2004]), the algorithm presented here exploits properties of the network topology to boost the performance of distributed query processing in the case of cycles. The discovery of the part of the PDMS network topology relevant to the query is done automatically during a first pass of query processing without the need for any specific topology discovery messages. The algorithm handles dynamic networks in which new nodes and mappings may appear during algorithm execution. A network change occurring during the execution of query processing routines does not cause additional message exchanges. The network may process in parallel several queries that are submitted to different nodes.

4.1 Distributed Query Processing

In the description of the algorithm, we use the convention that variables, constants and functions with a superscript always denote variables, constants and functions local to the node denoted by the superscript.

When a peer receives a query to answer, the query is processed locally by the peer itself using its own data, this first answer is immediately replied back to the node (or user) which issued the query, and then the query is reformulated and propagated to the relevant neighbour peers, according to the involved mapping rules. When a peer receives an answer from an acquaintance, it stores the data into the local database, by essentially materialising the view represented by the head of the involved mapping rule; in practice, it will use local GLAV processing to update the local database, as suggested in [Calvanese *et al.*, 2004]. The actual time when data are sent back may depend on the particular method of global query processing optimisation. Shorter delays may increase the number of messages in the network at global and local level but may decrease the time to get a complete answer (see last Section).

Each query q is labelled by a unique identifier id, which is assigned the first time the query is formulated. When a query q with identifier id is processed by a node, the node remembers that by storing the pair $\langle id, q\rangle$, in order to later understand when to stop in evaluating fixpoints. A query is propagated along the maximal dependency paths induced by the mapping rules, and, if the maximal dependency path contains a cycle, the query will reach a node with the same id, which is the node where the cycle is closed. In order to correctly compute the fixpoint induced by the presence of cycles, a node should continue to propagate queries to relevant neighbours along the maximal dependency paths until none of the answers from these propagated queries brings in new information. The following theorem states how to correctly decide when a peer has a complete query answer and the local query processing can stop.

Theorem 2 *The peer has a complete answer to a query if and only if the answers to all the reformulated queries propagated along all the maximal dependency paths do not bring new data for the query to the peer.*

There are $O((n-1)!)$ or $O((n-1)^{1/2}((n-1)/e)^{n-1})$ maximal dependency paths in a network of n peers. A naive checking of the condition of theorem 2 (as it was done in [Franconi *et al.*, 2004a]) would require exponential time and exponential number of messages wrt n. In this paper we introduce a way to restrict the conditions of theorem 2, in order to reduce the complexity to polynomial. All (exponentially many) possible paths can be enumerated as the combination of (polynomially many) *independent paths*[Diestel, 1997], the number of which is a cyclomatic number of the network[Diestel, 1997] and does not exceed n^2 . For example, in the example given in Section 2, the path $ABCDA$ can be considered as the combination of $ABCA$ and $ACDA$. It is easy to see that theorem 2 holds also if we consider only the independent maximal dependency path, reducing therefore the complexity as desired. As a matter of fact, this optimisation is very similar to the tabled execution of a datalog program[Chen and Warren, 1996].

Each node implicitly gets as part of a query answer explicit information about the topology of the sub-network it depends upon. During the first stage of its execution, the query processing algorithm at each node exploits the $AsynchSpanningTree$[Lynch, 1996] algorithm in order to construct a spanning tree of the graph. To implement

this, no change to the abstract behaviour of the algorithm is required, but only one additional field of the size of the identifier of the peer is required. Independent paths are built by exploiting the spanning tree. The overhead is not more then $E - n$ messages, where E is the number of involved mapping rules.

Please note that the addition or deletion of nodes does not change the properties of the algorithm: the neighbours of the added or deleted nodes will just consider the new queries/data from the added node or stop considering queries/data from the deleted node.

4.2 Handling of Existential variables

By exploiting results of [Calvanese *et al.*, 2004] we allow to use existential variables in the heads of the mapping rules. In order to simplify the processing of queries which use mappings with existential variables in the heads, we reduce query processing to use plain mapping rules without existential variables but with an extended set of relations.

The basic idea is based on skolemisation: for each rule $\exists y.q(x, y) \Leftarrow \exists z.q_b(x, z)$ we create a relation $ex_i(x, y)$, where i is the identifier of the mapping, and y is a new skolem constant, such that there is a one to one association between x and y (this can be easily generalised in case the arity of y is more than one). Then the mapping $\exists y.q(x, y) \Leftarrow \exists z.q_b(x, z)$ is rewritten as $q(x, y) \Leftarrow \exists z.q_b(x, z), ex_i(x, y)$. Then all mappings which use $q(x, y)$ must be rewritten such that new values are not propagated. It is assumed that we can determine for each constant if it is a new value without accessing to ex_i relations at different peers.

For example,

$\texttt{flight}(x, y), \texttt{flight}(y, z) \Leftarrow$
$\quad\texttt{two-leg-connection}(x, z)$

would be rewritten as

$\texttt{flight}(x, y), \texttt{flight}(y, x) \Leftarrow$
$\quad\texttt{two-leg-connection}(x, z), \texttt{ex}_1(x, z, y).$

In practice, the creation of a new relation is a not a good way to solve the problem since the relation may be large in the size of data (n^k, where n is the number of constants in the database, and k is the arity of the mapping rule), and most of the tuples of the relation are expected to be unused. We use a functional approach, i.e., each ex_i is implemented as a function which takes values x and returns the unique tuple representing y. The functional approach may actively use properties of the domain in order to easily generate a set of new values distinguished from a domain D.

4.3 The Algorithm

Data structures supported by each node:

1. id, an identifier of the node unique over the whole PDMS system.
2. $schema$, a relation describing all the relational data in a given node.

QUERY-NODE(Q)

```
1  key ← GENERATEUNIQUEVALUE
2  state[key] ← FALSE
3  RRules ← Rules where Rules[target]
   contains ATOMS(Q)
4  ▷ contains means that a collection contains
   at least one element of the set
5  ▷ Query propagation
6  for each rule in RRules
7      do
8          Query^{rule[target]}(key, rule[body])
9  ▷ Local computing of answer
   after distributed query processing
10 when complete[key] = TRUE
11     do
12         COMPUTEANSWER(Q)
```

Figure 1: The procedure Query-Node

3. $Rules(id, target, body, head)$, a relation describing all the mappings outgoing from a given node (which imports data). id is an identifier of the mappings, which is assumed to be unique across the network. $target$ is an identifier of the target of the mappings (node which exports data). $body$ is a string representing the body of the mapping as a conjunction of atoms from $schema^{target}$. $head$ is a string representing the head of the mappings as a set of atoms from $schema$.

4. $Paths(path)$, a relation describing all the paths beginning from a given node. Each $path$ is a string representing a conjunction of identifiers of mappings constituting the path.

5. $complete(key, state)$, is an array describing the completeness status of data received by a node during query evaluation.

6. $dcomplete(mapping, flag)$ is an array describing the state of the topology discovery. $dcomplete(mapping, \text{TRUE})$ means that the node knows the topology for all paths passing through the links $mapping$

7. $Queries(key, source, body, state)$, a relation which contains information about the queries being evaluated. key is the unique identifier of the query. $source$ is the identifier of the mapping through which the query was sent. $body$ is a string representing the query as a conjunctions of atoms. $state$ is a flag indicating the state of query execution, FALSE means that the query evaluation is not finished, TRUE means that the query evaluation is finished.

8. $SQueries(key, target)$, a relation describing the queries which were sent by a given node. key is the identifier of the query and $target$ is the identifier of the mapping through which the query was sent.

QUERY(key, $query$)

```
1   ▷ Query processing in intermediate node
2   source ← SOURCE()
3   if π_{key,source} Queries contains ⟨key, source⟩
4   ▷ If a request for a query identified by key
    was already received by mapping source
5       do
6           exit
7   Insert into Queries Values (key, source, query, FALSE )
8   QA ← COMPUTEANSWER(query)
9   state ← complete.state where complete.key = key
10  if Rules ≡ ∅
11      do
12          ▷ Stop topology discovery for leaf nodes
13          dstate ← COMPLETEFULL
14          dcomplete ← TRUE
15  if SQuery containsAll (Rules.key where Rules.head
    contains ATOMS(QUERY))
16      do
17          ▷ Stop topology discovery
    if there is no extension of a query path
    for a given node
18          dstate ← COMPLETEFULL
19          dcomplete ← TRUE
20  Answer^{Source(source)}(key, QA, ⟨id⟩, state, dstate)
21  RRules ← Rules where Rules[target] containsATOMS(query)
22  ▷ this part of algorithm is valid only
    if database does not contain constraints
23  for each rule in RRules
24      do
25          if π_{key,target} SRules contains ⟨key, rule[key]⟩
26              do
27                  next
28          Rewrite_Query^{Target(rule[key])}(key, rule[body])
29          insert into SQueries Values (key, rule[key])
```

Figure 2: The procedure Query

9. $EmptyUpdate(key, path, flag)$, a relation which logs the paths through which a query answer brings no update to the local data.

The following are the predefined functions we assume to be supported by each node:

1. GENERATEUNIQUEVALUE, a function which generates a unique string value across the whole network.
2. COMPUTEANSWER(Q),a function which computes answer to a query using only local data.
3. ATOMS(Q), a function which returns the set of atomic symbols in a query Q.
4. SOURCE(), a function which returns the identifier of the mapping through which the query was sent. Function is callable only from inside the function QUERY.
5. TARGET(), a function which returns the identifier of a mapping.

ANSWER(key, QA, $Path$, $state$, $dstate$)

```
1   ▷ Answer propagation
2   ▷ key is an identifier of the query
    to which answer is returned
3   ▷ QA is a dataset representing the query answer
4   ▷ Path is a sequence of identifiers of nodes participating
    in the computation of the answer
5   ▷ state is a flag for the completeness of the answer
6   ▷ dstate is a flag needed for topology discovery
7   mapping ← TARGET()
8   ▷ Information about network
9   ▷ Propagated only at the first stage of
    the execution of the query
10  if dstate = COMPLETE or dstate = COMPLETEFULL
11      do Insert into Paths values (Path)
12  if dstate = COMPLETEFULL
13      do dcomplete[mapping] ← TRUE
14  if for all rmapping in SQuery.mapping
    where SQuery.key = key dcomplete[rmapping] = TRUE
15      do
16          ▷ If a node knows all paths
    for all relevant incoming mappings
    then it knows full topology
17          dcomplete = TRUE
18  ▷ Processing of received answer
19  bEmptyUpdate ← UPDATELOCALDATE(QA)
20  if bEmptyUpdate = TRUE
21      do
22          Update set EmptyUpdates.flag = TRUE
    where EmptyUpdates.key = key
    and EmptyUpdates.path = Path
23    else
24      do
25          Update set EmptyUpdates.flag = FALSE
    where EmptyUpdates.key = key
    and EmptyUpdates.path = Path
26  ▷ Termination condition
27  if dcomplete[key] = TRUE and
    forall EmptyUpdates.flag = TRUE
    where EmptyUpdates.key = key
28      do
29          complete[key] ← TRUE
30  ▷ Answer propagation
31  for each rule in Queries where Queries.body
    contains ATOMS QA
32      do
33          QAS ← ComputeLocalAnswer(rule)
34          Answer^{Source(rule.rule)}
    (key, QAS, Path + id, state[key], dstate[key])
```

Figure 3: The procedure Answer

6. SOURCE(MAPPINGID),a function which given a mapping identifier returns the source node identifier (a source node is the node in the head of the mapping).
7. TARGET(MAPPINGID), returns the target identifier of the mapping.
8. $Rewrite_Query^{Target(rule[key])}(key,\ rule[body])$ is a rewriting algorithm, which takes a query, mappings

and generates maximal rewriting, then sends it to target source.

In order to get an answer to a query Q to a node i, the procedure QUERY-NODE(Q) of the node i should be called. The query answering algorithm is presented in Figures 1, 2, 3.

Since the algorithm is based on asynchronous messaging, a query node receives a continuous flow of answers, which are required to compute the query answer. It is easy to see that the system is monotonic, namely that at any moment in time during the execution of the query answering algorithm a query node always contain a sound – but possibly incomplete – answer to the query. The query answering algorithm terminates in a query node when the query node has the complete answer to the original query. This can be checked with the following termination condition.

Lemma 3 *A query node has a complete answer to a query if and only if there is no answer for that query through any incoming dependency path bringing new answer tuples to the query node.*

To check the termination condition each node checks the query answers it gets through each incoming dependency path. If some path did not bring any new answer tuples, the node "closes" that path locally. If all possible paths do not bring new answers, then a node declares that it has the complete answer to the query. In the case some path brings new data, then all dependency paths which have at least a node in common with the current query answer path are changed back to the state "open".

The problem of a naive implementation of the above termination condition is that in general there is an exponential number of dependency paths in the network, leading to an exponential number of query answer messages and exponential time of query processing with respect to the size of the network.

4.4 Independent Dependency Paths

The independent dependency paths optimisation is needed to speed up the performance of the algorithm in case of cyclic topologies, especially in the case of a large number of cycles. As explained above, a naive approach makes query processing time exponential in the number of nodes in the network. Experimental results published in [Franconi *et al.*, 2004b] have shown that even for small cyclic networks (e.g., cliques) and small datasets query processing becomes intractable very quickly.

The algorithm proposed in this article makes the message complexity of query processing not more then cubic in the number of nodes in the worst case.

An additional function INDEPENDENT_PATHS computes the set of *independent* dependency paths, which form a linear basis to describe all paths in the graph [Tutte, 1984], namely in the strongly connected component of the dependency graph. We use the property of directed graphs

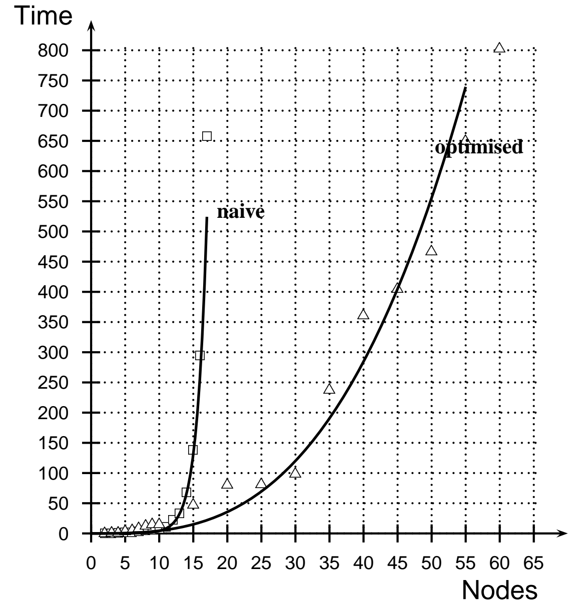


Figure 4: Query processing using Independent Paths vs. the Naive Approach: total time before termination for a clique network. Time is the total time of query execution in seconds. Nodes is the number of nodes across networks.

that any path in the graph is contained in a linear combination of some independent paths. We can show that the propagation of the query answer through all independent paths is equivalent to the propagation of the query answer through all the dependency paths in the graph.

A set of independent paths is stored in the relation *Independent_Path*. After the topology discovery phase, the node finds the set of independent paths and propagates it to intermediate nodes to create query answering paths which are stored in the relation $Answer_Propagation$ of each node. After query answering paths are propagated, each node propagates the query answer according to the independent paths only.

The number of independent paths does not exceed n^2 and the length of each path is less than n, where n is the number of nodes. This makes the communication complexity bounded by $O(n^3)$, with $O(n^2)$ being the time complexity of query processing over the network. We have to notice that the worst case n^3 holds only for highly connected networks with many cycles. In high scale Internet-like networks the number of independent cycles (the cyclomatic number of the graph) is around n and the average length of the cycle is around $log(n)$ [Newman, 2003; Gleiss *et al.*, 2000]. Research in the graph structure of the Internet and of web networks [Dill *et al.*, 2001] shows that despite the fact that, in general, network graphs are strongly connected (or better, that the strongly connected component of the network graph constitutes a big part of whole network), the network has a fractal cluster, which gives the opportunity for further topology optimisation. Independent paths maybe selected only in a branch of the current node. This leads to the cluster optimisation technique presented in the next Section.

4.5 The cluster optimisation

The cluster optimisation method tries to decrease the number of messages in the network, but still preserving the asynchronous communications to get complete answers. This optimisation was created for networks with self-similarity property and networks which consist of highly-connected components weakly-connected to each other. According to [Newman, 2003] this a common property of internet like networks.

The cluster optimisation method selects in distributed way a highly-connected subcomponent of the graph and tries to localise communications inside the component. Data are not transfered between components before fixpoint is not reached inside a component. After the fixpoint reached, the data are transfered between boundary peers.

The key intuition behind this method is that a cluster of peers (highly connected component) behaves as one peer for the outside network. During the first stage of the query propagation, after a cluster is identified, it is assigned a representing identifier which is equal to the minimal (in numerical or lexicographical order) id inside the cluster. During a query answering process each peer runs as it would have no edges going outside the cluster. After the fixpoint is reached by the boundary nodes, they propagate the query answer through the outgoing edges. The number of messages under asynchronous communications is expected to be in $\Sigma(m^3)$ times, where m is an average cluster size.

Our experiments with random graphs and "Internet-like" graphs has shown that for such kind of networks the expected time of query processing is $O(n\ log(n))$ for a small average out-degree of the nodes in the graph.

5 Evaluation

For our experiments we used traditional complexity measures for distributed algorithms [Lynch, 1996]:

- the *communication complexity* is the total number of messages throughout network before the algorithm terminates;
- the *time complexity* is the number of rounds before the algorithm terminates.

and the *total time* from *wakeup* to termination of the algorithm (similar to the so called *time to completion*). In all experiments measuring time, wall-clock time is reported.

The experiment environment consists of 4 PCs with Intel Pentium 4(R) CPU 1.70GHz, 512MB RAM and 900 MHz PowerPC G3, 640 MB. One of PCs is running Windows XP, and others are running Fedora Linux (Core 1). All computers were located inside a local network with bandwidth 100MHz. Communications between nodes are implemented in Java RMI protocol; we used a raw protocol instead of JXTA pipes, or JMS messages in order to make query processing as much efficient as possible, since most messages over networks are "Answer" messages. Currently JXTA is used only for node advertisement and to bind the

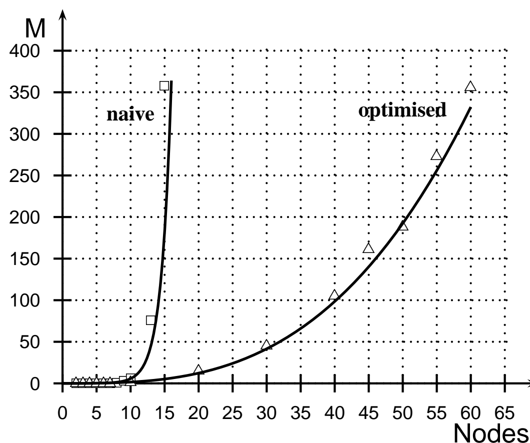


Figure 5: Query processing using Independent Paths vs. the Naive Approach: communication complexity for a clique network. M is the total number of messages divided on 500. Nodes is the number of nodes accross network.

name of nodes to their physical network location. Peer-to-peer database agents were executed with Java HotSpot Client VM J2SE 1.4.2 platform for PC computers and PowerPC. As a data-store, Oracle 10.1.0.2 deployed on a Windows XP platform was used. Agents belonging to one computer were running as threads in one single Java virtual machine.

PDMS nodes were modeled as threaded JAVA applications. Up to 20 logical nodes were run on one server. At a preliminary stage of the experiment it was discovered that up to 20 logical nodes may run one one physical server without interaction on experimental datasets. Before the experiment each node was loaded into the main memory with mappings and information about physical address of its adjacent nodes. The global discovery of the network topology at logical level was performed during experiment.

Six different relational schemas were used describing bibliographic data extracted from the DBLP and bibliographic database. Each database consists of 200 records about publications and a varying number of records (from 100 to 400) about properties of articles (authors, editors, publishers, journals). Scheme consisted of up to 6 relations with up to 6 non-key attributes. Mappings consisted of conjunctive queries up to conjunction of 3 atoms in the body and up to conjunction of two atoms in the head. Existential variables were used both in the body and in the header of the mappings.

All peer-to-peer agents were loaded into main memory with a complete configuration (mappings, logical and physical addresses of other agents) and with the connections already established before the starting of the experiments.

To model the behaviour of a dynamic network, a software simulation of the network changes was used: each agent has an interface to halt the agent with a notification to the neighbour agents.

As noticed before, the original naive version of the algorithm presented in the literature was based on a procedure

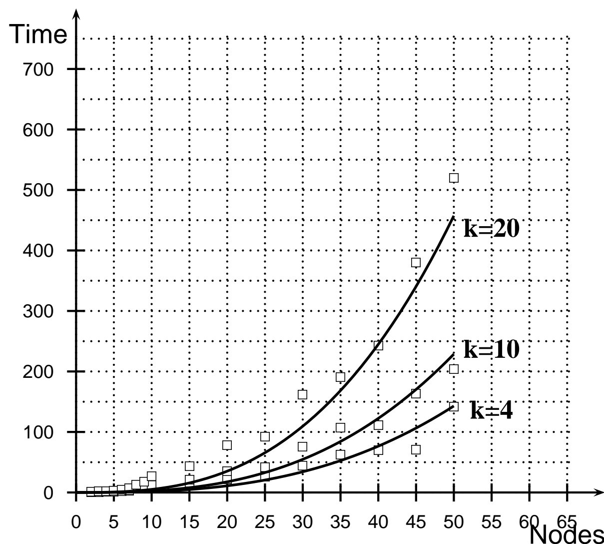


Figure 6: Independent Paths optimisation for "Internet like" network: total time before termination. k is the parameter of density of the network. For big n, k is the average number of outgoing links for a peer. Time is the total time of query processing in seconds. Nodes is the number of nodes accross network.

checking all possible node sequences in the dependency paths. This leads to exponential time of query processing with respect to the node-size of the network. One of the objectives of this paper is to test the *graph-optimisation* of query processing, i.e., how the properties of the network topology, namely the independent paths, may be used to make query processing more efficient, especially in large networks.

The charts in Figures 4 and 5 demonstrate that in highly-connected networks (experiments were performed for clique shaped PDMS networks) with many nodes the query processing utilising linear decomposition of the graph is much more efficient than "naive" implementation.

The "Independent Paths" optimisation is developed for large scale networks with high-level of connectivity inside the network. In acyclic networks or networks with very few cycles with respect to the number of nodes such optimisation decreases the efficiency of query processing due to the overhead in the number of messages and computations.

The cluster optimisation technique is also based on the utilisation of the graph structure of the PDMS network in order to reduce communication complexity. This technique attempts at localising the communications inside the small highly connected parts of the network. During the computation, the network is divided into clusters, which are highly-connected sets of nodes. Query processing is localised inside a cluster up to the reaching of a fix-point at the boundary nodes of the cluster. After that, data are transferred from cluster to cluster. As it was discovered in [Dill *et al.*, 2001], the Internet network has a fractal structure: it consists of subnetworks with a high-degree of connectivity within each of such subnetworks and low connectivity between subnetworks.

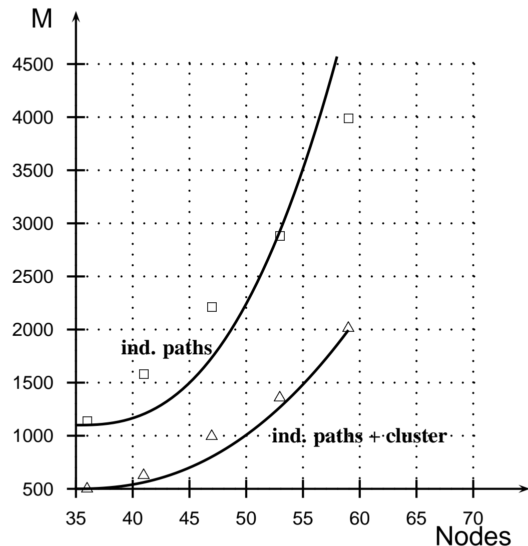


Figure 7: Cluster optimisation. M is the total amount of messages, Nodes is the total number of nodes across network.

For this set of experiments a network was created as a set of interconnected clusters. Each cluster is a clique of 6 nodes. Clusters are organised into a ring such that there is only one link between two connected clusters. The outcome of the experiments is shown in Figure 6 and 7.

6 Conclusions

In this paper we investigate different optimisation techniques for query processing in networks of autonomous sources. We have shown that the knowledge of the network structure may help to significantly improve the efficiency of query processing both in the number of exchanged messages and in the time to get a complete answer. Our methods of optimisation do not require the knowledge in advance of a predefined topology, since the network topology is discovered during query processing, even in the case of dynamic networks. In this paper we propose a notion of soundness and completeness of a query answer in the case of dynamic networks. The traditional definition proposed in distributed databases literature does not suit the context of PDMS networks, since different notions of data objects and query answers are used. The query answering algorithm is efficient with respect to changes of the network. The time of query processing in a network, when the size of the change during query processing is sensibly less than the size of the network is comparable to the time of query processing in a stable network. The assumption that a change is small with respect to the size of whole network is reasonable for large scale networks.

References

[Bernstein *et al.*, 2002] P. Bernstein, F. Giunchiglia, A. Kementsietsidis, J. Mylopoulos, L. Serafini, and I. Zaihrayeu.

Data management for peer-to-peer computing: A vision. In *Workshop on the Web and Databases, WebDB 2002*, 2002.

[Calvanese *et al.*, 2004] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Logical foundations of peer-to-peer data integration. In *Proc. of the 23rd ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS-2004)*, 2004. To appear.

[Chen and Warren, 1996] Weidong Chen and David S. Warren. Tabled evaluation with delaying for general logic programs. *Journal of the ACM*, 43(1):20–74, 1996.

[Crespo and Garcia-Molina, 2002] Arturo Crespo and Hector Garcia-Molina. Routing indices for peer-to-peer systems. In *Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02)*, page 23. IEEE Computer Society, 2002.

[Daswani *et al.*, 2003] Neil Daswani, Hector Garcia-Molina, and Beverly Yang. Open problems in data-sharing peer-to-peer systems. In *ICDT 2003*, 2003.

[Diestel, 1997] Reinhard Diestel. *Graph Theory*. Springer, New York, 1997.

[Dill *et al.*, 2001] Stephen Dill, Ravi Kumar, Kevin McCurley, Sridhar Rajagopalan, D. Sivakumar, and Andrew Tomkins. Self-similarity in the web. In *Proc. of Very Large Data Bases (VLDB 01)*, pages 69–79, 2001.

[Fagin *et al.*, 2003] Ronald Fagin, Phokion G. Kolaitis, R. J. Miller, and Lucian Popa. Data exchange: Semantics and query answering. In *Proceedings of the 9th International Conference on Database Theory*, pages 207–224. Springer-Verlag, 2003.

[Franconi *et al.*, 2003] Enrico Franconi, Gabriel Kuper, Andrei Lopatenko, and Luciano Serafini. A robust logical and computational characterisation of peer-to-peer database systems. In *Proceedings of the VLDB International Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P'03)*, 2003.

[Franconi *et al.*, 2004a] Enrico Franconi, Gabriel Kuper, Andrei Lopatenko, and Ilya Zaihrayeu. A distributed algorithm for robust data sharing and updates in p2p database networks. In *Proceedings of the EDBT International Workshop on Peer-to-peer Computing and Databases (P2P&DB'04)*, March 2004.

[Franconi *et al.*, 2004b] Enrico Franconi, Gabriel Kuper, Andrei Lopatenko, and Ilya Zaihrayeu. Queries and updates in the coDB peer to peer database system. In *Proc. of the 30th International Conference on Very Large Data Bases (VLDB'04)*, 2004.

[Garcia-Molina *et al.*, 1997] Hector Garcia-Molina, Yannis Papakonstantinou, Dallan Quass, Anand Rajaraman, Yehoshua Sagiv, Jeffrey D. Ullman, Vasilis Vassalos, and Jennifer Widom. The TSIMMIS approach to mediation: Data models and languages. *Journal of Intelligent Information Systems*, 8(2):117–132, 1997.

[Gleiss *et al.*, 2000] Petra M. Gleiss, Peter F. Stadler, Andreas Wagner, and David A. Fell. Small cycles in small worlds. arxiv.org, Sep 2000. preprint.

[Halevy *et al.*, 2003] Alon Halevy, Zachary Ives, Dan Suciu, and Igor Tatarinov. Schema mediation in peer data management systems. In *Proceedings of the 19th International Conference on Data Engineering (ICDE'03)*, 2003.

[Holliday *et al.*, 2003] JoAnne Holliday, Robert C. Steinke, Divyakant Agrawal, and Amr El Abbadi. Epidemic algorithms for replicated databases. *IEEE Trans. Knowl. Data Eng.*, 15(5):1218–1238, 2003.

[Kementsietsidis and Marcelo Arenas, 2003] Anastasios Kementsietsidis and Renée Miller Marcelo Arenas. Managing data mappings in the hyperion project. In *Proceedings of the 19th International Conference on Data Engineering (ICDE'03)*, 2003.

[Kementsietsidis *et al.*, 2003] Anastasios Kementsietsidis, Marcelo Arenas, and Renee J. Miller. Mapping data in peer-to-peer systems: Semantics and algorithmic issues. In *Proceedings of the SIGMOD International Conference on Management of Data (SIGMOD'03)*, 2003.

[Kirk *et al.*, 1995] T. Kirk, A. Y. Levy, Y. Sagiv, and D. Srivastava. The Information Manifold. In C. Knoblock and A. Levy, editors, *Proceedings of the AAAI 1995 Spring Symp. on Information Gathering from Heterogeneous, Distributed Environments*, Stanford University, Stanford, California, 1995.

[Lenzerini, 2002] Maurizio Lenzerini. Data integration: a theoretical perspective. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 233–246. ACM Press, 2002.

[Lynch, 1996] Nancy A. Lynch. *Distributed Algorithm*. The Morgan Kaufmann Series in Data Management Systems. Morgan Kaufman Publishers, Inc., 1996.

[Newman, 2003] M. E. J. Newman. The structure and function of complex networks. *SIAM Review*, 45(2):167–256, 2003.

[Ratnasamy *et al.*, 2001] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. In *Proceedings of the ACM SIGCOMM '01 Conference*, San Diego, California, August 2001.

[Serafini and Ghidini, 2000] Luciano Serafini and Chiara Ghidini. Using wrapper agents to answer queries in distributed information systems. In *Proceedings of the First Biennial Int. Conf. on Advances in Information Systems (ADVIS-2000)*, 2000.

[Serafini *et al.*, 2003] Luciano Serafini, Fausto Giunchiglia, John Mylopoulos, and Philip A. Bernstein. Local relational model: A logical formalization of database coordination. In *CONTEXT 2003*, pages 286–299, 2003.

[Stoica *et al.*, 2001] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM '01 Conference*, San Diego, California, August 2001.

[Tatarinov and Halevy, 2004] Igor Tatarinov and Alon Halevy. Efficient query reformulation in peer data management systems. In *Proceedings of the SIGMOD International Conference on Management of Data (SIGMOD'04)*, 2004.

[Tutte, 1984] W. T. Tutte. *Graph Theory*, volume 21 of *Encyclopedia of Mathematics and its Applications*. Addison-Wesley Publishing Company, 1984.