# GrouPeer: Dynamic Clustering of P2P Databases

Verena Kantere[1], Dimitrios Tsoumakos[2] Timos Sellis[1], and Nick Roussopoulos[2]

[1] School of Electr. and Comp. Engineering, National Technical University of Athens,
{vkante, timos}@dbnet.ece.ntua.gr,
[2] Department of Computer Science, University of Maryland, College Park,
{dtsouma, nick}@cs.umd.edu

**Abstract.** Sharing structured data in a P2P network is a challenging problem, especially in the absence of a mediated schema. The standard practice of answering a consecutively rewritten query along the propagation path often results in significant loss of information. On the opposite, the use of mediated schemas requires human interaction and global agreement, both during creation and maintenance. In this paper we present *GrouPeer*, an adaptive, automated approach to both issues in the context of unstructured P2P database overlays. By allowing peers to individually choose which rewritten version of a query to answer and evaluate the received answers, information-rich sources left hidden otherwise are discovered. Gradually, the overlay is restructured as semantically similar peers are clustered together. Experimental results show that our technique produces very accurate answers and builds clusters that are very close to the optimal ones by contacting a very small number of nodes in the overlay.

## 1 Introduction

In the last few years, there has been a growing interest in the Peer-to-Peer (P2P) paradigm, primarily boosted by popular applications that enable massive data sharing among millions of users. The P2P paradigm dictates a fully distributed, cooperative network design, where nodes collectively form a system without any supervision. Many popular P2P applications operate on unstructured networks, with peers joining and leaving the system in an ad-hoc fashion, while maintaining only local knowledge. While structured overlays (e.g., [40]) provide efficient lookup operations, in many realistic scenarios the topology cannot be controlled and thus they cannot be used (e.g., dynamic ad-hoc networks or existing large-scale unstructured overlays).

In contrast to data integration architectures, P2P data sharing systems do not assume a mediated schema to which all sources of the system should conform in order to share data. In such a system, where peers share (semi)-structured data, each is an autonomous source that has a local schema. Sources store and manage their data locally, revealing only part of their schemas to the rest of the peers. Due to the lack of global schema, they express and answer queries based on their local schema. In a P2P data management system, peers also perform local coordination with their acquaintees, i.e. their one-hop neighbors in the overlay. Thus, both data management and coordination are performed in a totally decentralized manner. During the acquaintance procedure, the two peers exchange information about part of their local schema and create a mediating mapping semi-automatically [21]. The establishment of an acquaintance implies an agreement for the performance of data coordination between the acquaintees based on the respective schema mapping. However, peers do not have to conform to data or schema transformation in order to establish acquaintances with other peers and participate in the system.

In large-scale unstructured P2P systems as described above, joining peers usually become acquainted to the first randomly available nodes and not to the most useful ones, i.e., the peers that best meet their need for information. Therefore, they have to direct queries not only to their neighbors, but to a greater part of the system. Furthermore, the lack of global knowledge deprives peers from the ability to direct their queries to appropriate remote nodes. One can roughly identify two common approaches to this problem. A possible solution is to propagate queries on paths of bounded length in the overlay. At each routing step, the query is rewritten to the schema of its new host based on the respective acquaintance mappings. A query may have to be rewritten several times from peer to peer till it reaches nodes that are able to answer it sufficiently in terms of quality, but also quantity, of the result. It is obvious that the successive rewritings decrease or restrict the information that can be returned by a query and, thus, also reduce the possibility of accurate query answering. Moreover, it is the case that peers may not be able to sufficiently answer received queries, not because their local schema does not match the initial query adequately, but because the incoming rewritten version has been gradually reduced or corrupted. Therefore, the performance of the query processing procedure is degraded during the rewritings on intermediate peers.

In the second approach, nodes are organized by means of a human-guided process (usually by one or more administrators and application experts) into groups of peers that store semantically related data. The administrator, using schema matching tools as well as domain knowledge, creates a mediated schema representative of the group and mappings with the local databases. Queries are then expressed on this mediated schema. Obviously, this approach requires manual work, extensive peer coordination and repetition of this process each time the group changes.

**Motivating Example**

Envision a P2P system where the participating peers are databases of private doctors of various specialties, diagnostic laboratories and databases of hospitals. Figure 1 depicts a small part of this system, where nodes are: DavisDB - the database of the private doctor Dr. Davis, LuDB - the database of pediatrician Dr Lu and StuartDB - the database of the pharmacist, Mr Stuart. On top of each database sits a P2P layer, which is responsible for all data exchange of this peer with its acquaintees. Among others, the P2P layer is responsible for the creation and maintenance of mappings of local schemas during the establishment of acquaintances along the lines of [21]. Moreover, each peer owns a query rewriting and a query-schema matching mechanism. The schemas of the databases are shown in Figure 1.

Suppose that Dr Davis would like to collect from the system general information about patients that have had diseases. He expresses the following query on his database: $Q_{orig}$:

```
SELECT  V.Pid, D.DisDescr, D.Ache,
        T.Drug, T.Dosology
```

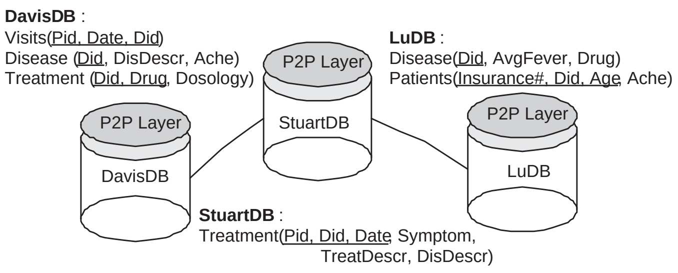

DavisDB :
Visits(Pid, Date, Did)
Disease (Did, DisDescr, Ache)
Treatment (Did, Drug, Dosology)
LuDB :
Disease(Did, AvgFever, Drug)
Patients(Insurance#, Did, Age, Ache)
P2P Layer
StuartDB
P2P Layer
DavisDB
P2P Layer
LuDB
StuartDB :
Treatment(Pid, Did, Date, Symptom,
TreatDescr, DisDescr)

**Fig. 1.** Part of a P2P system with peer-databases from the health environment

```
FROM    Disease D, Treatment T, Visits V
WHERE   V.Did = D.Did AND D.Did = T.Did
```

Having only one acquaintance, the pharmacist's database, Dr. Davis's database propagates $Q_{orig}$ to it. We assume GAV, LAV, or GLAV (i.e. Global, Local, Global and Local As View) mappings between acquaintees [23]. Note that the 'directionality' of the mapping from one peer to another characterizes the same mapping as GAV or LAV. The directionality of a mapping is decided with respect to the direction of the query rewriting between two peers. Thus, a mapping between peers $P_1$ and $P_2$ can be used for LAV rewriting (thus, a LAV mapping) from $P_1$ to $P_2$ and, also, it can be used for GAV rewriting (thus, a GAV mapping) from $P_2$ to $P_1$. We assume the following LAV mapping between DavisDB and StuartDB databases:
$M_{StuartDB\_DavisDB}$:
Treatment$_{StuartDB}$(Pid, _, _, Symptom, TreatDescr, DisDescr):-Visits$_{DavisDB}$(Pid, _, Did), $Disease_{DavisDB}$(Did, DisDescr, Ache), Treatment$_{DavisDB}$(Did, Drug, _), $\{Symptom = Ache, TreatDescr = Drug\}$
where the correspondences Symptom = Ache, TreatDescr = Drug that are implied are added in a set at the end of the mapping. [3] Thus, the rewritten query on StuartDB is the following:
$Q_{StuartDB\_sr}$:

```
SELECT  T.Pid, T.DisDescr, T.Symptom, T.TreatDescr
FROM    Treatment T
```

Obviously the new query has lost the attributes referring to information about drug dosology, since it cannot be mapped in StuartDB. The node of Mr Stuart passes the rewritten version $Q_{StuartDB\_sr}$ to Dr Lu with whom he has the following GAV mapping:
$M_{StuartDB\_LuDB}$:

[3] The mapping is actually a view defined on StuartDB.Treatment, which is matched with a join on DavisDB relations such as:
View1(Pid, Symptom, TreatDescr, DisDescr):-Treatment(Pid,Did, Date, Symptom, TreatDescr, DisDescr)
View1(Pid, Ache, Drug, DisDescr):- Visits(Pid, Date, Did),Disease(Did, DisDescr, Ache), Treatment(Did, Drug, Dosology)
Due to lack of space we summarize mappings by omitting view definitions and introducing '_' for attributes that are not needed.

$\text{Treatment}_{StuartDB}$(Pid, _, _, Symptom, _, _):- $\text{Disease}_{LuDB}$(Did, AvgFever, _), $\text{Patients}_{LuDB}$ (Insurance$\sharp$, Did, _, _), Age $< 13$, $\{Pid = Insurance\sharp, Symptom = AvgFever\}$
where correspondences Pid = Insurance$\sharp$, Symptom = AvgFever that are implied are added in a set at the end of the mapping. Thus, the rewritten query on LuDB is the following:
$Q_{LuDB\_sr}$:

```
SELECT  P.Insurance#, D.AvgFever
FROM    Disease D, Patients P
WHERE   D.Did = P.Did, P.Age < 13
```

Obviously the new query has lost more attributes, which refer to the description of the disease and the respective drug. Moreover, the new query is more restrictive than the original, since it has an additional condition on 'Age'. Finally, it is clear that the 'Ache' attribute of the original query has been poorly rewritten to 'AvgFever', even though the schema of LuDB contains an attribute that represents the exact same concept. Yet, if Dr Davis were acquainted with Dr Lu, among the supported mappings could be:
$M'_{DavisDB\_LuDB}$:
$\text{Visits}_{DavisDB}$(Pid, _, Did), $\text{Disease}_{DavisDB}$ (Did, _, Ache), $\text{Treatment}_{DavisDB}$ (Did, Drug, _):-$\text{Disease}_{LuDB}$(Did, _, Drug), $\text{Patients}_{LuDB}$(Insurance$\sharp$, Did, _, Ache), $\{Pid = Insurance\sharp\}$
Using the above mapping, Dr Davis would ideally like his query to be translated as follows:
$Q_{LuDB\_ideal}$:

```
SELECT  P.Insurance#, D.Ache, D.Drug
FROM    Disease D, Patients P
WHERE   D.Did = P.Did
```

Apparently, the above rewritten version overcomes the degradation of successive rewriting in terms of query information loss and further query restriction, as well as the poor matching of the 'Ache' attribute.

In the proposed framework (*GrouPeer*), DavisDB can evaluate Dr Lu's query translations (e.g., suggest that 'Ache'='AvgFever' is not a good correspondence and 'Pid'='Insurance' is a good one) and enable him to gradually improve the quality of its query rewriting. Through mutual iterative evaluations Dr Davis notices the average answer quality from Dr Lu is high enough to add him as an acquaintee. The two nodes create complete mappings between their schemas, a task that *GrouPeer* greatly facilitates by building on mappings formed during remote query processing. The details on this example are in Section 3.5.

**Our Proposal: GrouPeer**

The above example points out one of the major problems in unstructured P2P database systems: peers may not be able to obtain requested information or learn about peers with similar interests because of insufficient schema similarity between acquaintees. Semantic grouping that would assist in this task requires manual coordination at each group creation/maintenance event. In this work we describe *GrouPeer*, a system designed to enable accurate query evaluation through semantic overlay clustering and automatic creation and maintenance of semantic groups in relational P2P databases without prior schema or meta-schema information. In *GrouPeer*, nodes individually decide whether to answer the successively rewritten query or automatically rewrite its original version. Requesters evaluate the replies along with the returned rewritings and gradually build mappings with remote peers. Eventually, peers with similar local schemas become acquainted and clusters are created around active peers. Our paper makes the following contributions:

- Adapts classical query rewriting to the needs of P2P databases and combines it successfully with automatic schema matching.
- Investigates the notion of semantic query similarity in the context of the P2P paradigm and proposes directions for its quantification.
- Presents a complete methodology for discovering similar peers in an unstructured overlay and gradually clustering them by utilizing learning through regularly posed query traffic.
- Exploits the results of learning so that mappings between remote peers are gradually built on their specific common interests; this facilitates the acquaintance procedure that is usually performed through human interaction.

Our experimental section shows how clustering efficiently reorganizes any given overlay so that peers can direct queries to relevant nodes and increase answer accuracy. We then show how grouping can be applied to increase both the accuracy and the number of received replies.

The rest of the paper is organized as follows: Section 2 discusses aspects of the proposed clustering technique. In Section 3 we discuss aspects of query similarity and in Section 4 we analyse the query reformulation procedure. In Section 5 we present formally the clustering process. Section 6 shows experimental results and Section 7 presents related work. Finally, Section 8 summarizes our work.

## 2 Discovering Remote Interesting Peers

As our motivating example demonstrated, the querying node is doomed to ignorance of the information-rich peers because of enforced reformulation of queries on each node of the propagation path. But what if these nodes had the chance to receive and answer the originally posed query? Then, the inquiring node would probably (a) get better answers to its query, (b) have the chance to learn about peers with similar information, (c) get acquainted with them and get even better query answers.

In *GrouPeer*, we propose a procedure that supports the evasion of successive rewritings on every peer of a query's propagation path, instead of, sometimes hopelessly, refining query reformulation. This methodology enables peers to discover others with similar interests and schemas, that cannot be tracked otherwise. Pairs of remote peers that exchange queries and answers learn gradually about the schema of the other party. Learning is performed through making queries and evaluating their answers, and is formed in mappings between the schemas of the two peers. These mappings encapsulate the common interest of the two peers, since they refer to the vital schema parts on which they express and answer queries. If the peers decide to become acquainted, these

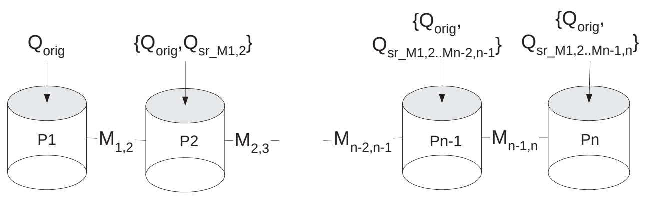

$Q_{orig}$
$\{Q_{orig}, Q_{sr\_M1,2}\}$
$\{Q_{orig}, Q_{sr\_M1,2..Mn-2,n-1}\}$
$\{Q_{orig}, Q_{sr\_M1,2..Mn-1,n}\}$
P1
$M_{1,2}$
P2
$M_{2,3}$
$M_{n-2,n-1}$
Pn-1
$M_{n-1,n}$
Pn

**Fig. 2.** Propagation of a query $Q_{orig}$ posed on peer $P1$ along a path of $n$ acquainted peers. Each peer receives a pair of queries: one is $Q_{orig}$ itself and the other is the version of $Q_{orig}$ that is successively rewritten through the chain of mappings.

mappings are already a language for their communication and alleviate the administrator's load for creation of mappings for the new acquaintance. Overall, the proposed methodology leads to the gradual clustering of the P2P system in groups with common interests. Thus, we refer to it as the 'clustering process'.

In the following, we present a brief overview of the clustering process in order to give the reader a flavor of what's coming next. Moreover, we identify interesting subproblems that we solve while developing the peer clustering method. Specifically, we make necessary clarifications about the role of query reformulation and query similarity in our approach.

## 2.1 Overview of the Clustering Process

In order to achieve the discovery of remote relevant peers, the key idea of our method is to propagate along the query path not only the successively rewritten version, but also the original one. In this way, the peers receiving this pair of query versions can individually decide which one to answer. Peers are assumed to be equipped with a query rewriting mechanism and an automatic schema-matching tool. The rewriting mechanism is used in order to rewrite queries expressed on schemas of acquaintees based on the respective mappings. The automatic schema-matching tool is used in order to comprehend and translate queries or part of queries expressed on schemas for which mappings are not available.

Successive query reformulation produces query versions that deviate from the original query. Obviously, if the chain of peer mappings used for the rewriting is poor in information relevant to the query (i.e. query parts cannot be reformulated accurately), this can result in fast degradation within a few hops. Query parts that cannot be translated through existing mappings are eliminated in the rewritten version. Even if the following nodes on the query path encapsulate the eliminated concepts in their schemas, they still cannot contribute them to the original query, because the version they receive does not include them. Our goal is to keep the eliminated concepts aside and try to match them in follow-up schemas.

Overall, an initiated query $Q_{orig}$ is propagated in the query path. On each node, the query is rewritten through mappings with the previous node to $Q_{sr}$, which is augmented with automatically rewritten query parts to $Q_{sra}$. Also, $Q_{orig}$ is automatically rewritten from scratch to $Q_{ar}$. The answering node compares the two rewritten versions with

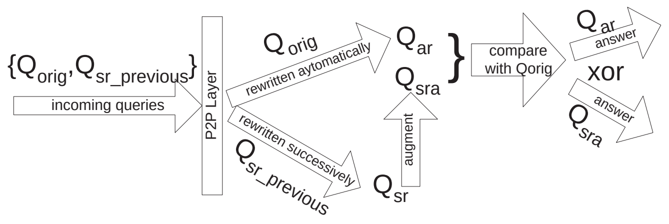


**Fig. 3.** Representation of the query answering procedure on a peer.

the original one, using a special similarity function, (see Section 3), and answers the version it deems most similar to it. Figure 2 shows a propagation path of a query and Figure 3 summarizes the main part of the query answering procedure on a peer. The query initiator evaluates the satisfiability of the received answer and sends its feedback to the answering peer about the answered query version. According to the evaluation, the query replier keeps record of bad and good rewritings of the initiator's schema elements. Gradually, the query replier builds mappings with the initiator through the queries it receives and answers on its behalf. Moreover, the initiator logs the evaluation of query answers from each replier. Based on this, the initiator can decide that it has common interests with a remote peer and ask to become its acquaintee. After that, new acquaintees can base their communication on already created mappings.

## 2.2 Query Similarity and Query Reformulation

We are interested in data exchange issues in pure (i.e. without super-nodes) P2P database systems. We consider peers that each one of them owns a relational peer schema (i.e., the only internal mappings are foreign key constraints) that it thoroughly exports to its immediate neighbors, hereafter *acquaintees*. Each pair of acquaintees holds peer mappings between their schemas. Peer mappings are considered to be of the well-known GAV-LAV-GLAV (i.e. Global, Local, Global and Local As View) form (we limit our study to mappings that can be expressed as SPJ queries). Hence, a peer mapping is a view with the head of it belonging to the global schema and the body to the local one (GAV) or the opposite (LAV). For clarity reasons, we remind that in GAV/LAV definitions for the P2P setting, the global schema is the schema of the peer on which the query is initially posed and the local schema is the schema of the peer on which it is rewritten. Peers express queries on their local schema, which are then propagated in the P2P system from one acquaintee to the next. We focus on conjunctive queries with arithmetic comparisons. At each step, a query is successively reformulated through mappings.

In such a system we assume that the query reformulation is performed by a query rewriting mechanism based on the chain of algorithms for answering queries using views [25], [30], [35] and [3] for LAV mappings; for GAV mappings, we consider the straightforward substitution of the view head with the body as it is done in Piazza [17]. GLAV mappings are used with combination of LAV and GAV query rewriting techniques [13, 17]. However, we propose that the query reformulation mechanism is enhanced, so that queries are rewritten even if they can be partly satisfied by the available mappings. More specifically, the reformulation mechanism needs to address the cases where partial transformation of the queries is possible; i.e. not all ’select’ attributes or ’where’ conditions are present in mappings. In the following we describe the characteristics of such a mechanism in detail.

The goal of the reformulation mechanism is to transform a query so that it can be answered partly or thoroughly by an acquaintee, i.e. an one-hop neighbor in the overlay. The available query rewriting algorithms restrict their usage to queries that can be completely rewritten under a set of mappings, meaning that these algorithms can rewrite queries only if all ‘select’ attributes and ‘where’ conditions of the original query can be rewritten through the available mappings.

Yet, this is not suitable for a P2P environment. In many such common applications, peers are satisfied with retrieved information with characteristics similar to those of their query (consider for example search engines, popular P2P file-sharing applications, etc). Therefore, it is reasonable to assume that it is preferable for our P2P database system to operate in a similar manner (as [33] does). Hence, we would like peer queries to be reformulated and propagated even if they can be only partly satisfied.

Existing rewriting algorithms have been designed to serve the problem of data and schema integration and thus do not allow partial query rewriting. These algorithms are driven by the assumption that the correct rewriting of a query is the maximally-contained version of it [24]. Yet, query containment in data integration refers to the containment of the answer sets. In P2P database environments, peers are not interested in answers from other peers that are contained in the answers they can retrieve locally; they are rather interested in answers that would be ”semantically relevant” to the originally posed query.

In this work we propose the preprocessing of the queries posed on P2P databases in order to produce versions that can be classically rewritten to other peer schemas. In this context, we investigate the notion of query similarity based more on structural features rather than semantics itself and we propose a query similarity function . In the same spirit we discuss guidelines for the preprocessing of queries and we present an algorithm that considers GLAV, GAV and LAV mappings in order to select those that can rewrite the query in the best way.

GrouPeer gives the opportunity to overcome peers on a query propagation path that are poor in requested information and discover others that are rich. The trade-off for this opportunity is the sacrifice of soundness and completeness of query answers. Also, each peer evaluates the soundness and completeness of its answered query versions from its own perspective using its own tools that can extend from basic dictionaries to sophisticated automatic matches that emply ontologies. Nevertheless, *GrouPeer* allows the peer users to define their own preference for soundness and completeness through the weights on query elements and the similarity threshold.

## 3 Query Similarity

In order for a peer to answer an incoming query, it has to translate it with respect to its local schema. Usually, the resulting query is not a complete translation of the original one, but is somehow ‘similar’ to it. In the literature [24], the similarity between rewritten queries is measured according to the containment of the results of the rewritten query to the results of the original one. However, this kind of query similarity cannot be effective in the context of this work. The difference of the P2P database paradigm and the data integration one, is that the first is multi-layered whereas the second has only two layers. This means that a query has to be rewritten several times along a propagation path (i.e. many layers of rewriting) in a P2P system whereas it is rewritten only once in a data integration system. In a P2P system the composition of a chain of mappings between two remote peers is not known to either of them. In a P2P database system we would like to compare queries written on remote peer schemas: $Q_{orig}$ is written on the schema of the initiator and the rewritten version $Q_{rewr}$ on the schema of a remote peer. Moreover, in *GrouPeer* the $Q_{rewr}$ is not a classically rewritten version of $Q_{orig}$. For these two reasons, it is not possible to decide query containment based on view expansion. Furthermore, we require a quantification of such a containment, which is not defined in a classical way. Nevertheless, query containment in data integration is actually decided by the containment of the query answers. This means that if we actually want to measure the classical containment of $Q_{orig}$ and $Q_{rewr}$, we need the query answers, (since it is not possible to decide query containment based on view expansion). However, our goal is not to answer the queries in order to measure their similarity, but to measure their similarity in order to decide which one to answer. Thus, we have to rely on query characteristics rather than the answer to the query in order to determine similarity.

Query similarity has been explored in several works in the recent past. Some of these works deal with keyword matching in the database environment [4, 7] or with the processing of imprecise queries [14, 22, 31]. The work in [5] talks about attribute similarity but focuses on numeric data and on conclusions about similarity that can be deduced from the workload. Furthermore, in [15] queries are classified according to their structural similarity; yet, the authors focus on features that differentiate queries with respect to optimization plans. The only work relevant to ours is that of [6], where overall semantic similarity of queries is explored. Yet, our focus is on query versions that are produced through the use of mappings, and we are interested on the effect of the mappings in query similarity.

In order to measure the similarity of two queries, $Q_{orig}$ and $Q_{rewr}$ there is a need for a function that quantifies their semantic relativeness, $M_{sim}(Q_{orig}, Q_{rewr})$. Our goal is to study from a qualitative point of view what is the impact on the query semantics of query elements that cannot be rewritten though the available mappings. Next, we discuss the guidelines along which the similarity function should be constructed and the factors that affect it.

## 3.1 Aspects of Query Similarity

The similarity of two queries, each on a distinct schema, is not only a matter of different query characteristics, but also of why these different characteristics exist. Specifically, since we are interested in incomplete rewritings of queries, query similarity has to take into consideration under which data exchange conditions new elements in the rewritten version are inserted or old elements are missing.

**(1) 'select' attributes:**

For example, remember the query on StuartDB, $Q_{StuartDB\_sr}$, and the successively rewritten version on LuDB, $Q_{LuDB\_sr}$; First, two of the 'select' attributes of $Q_{StuartDB\_sr}$, T.DisDescr and T.TreatDescr are missing in the 'select' clause of $Q_{LuDB\_sr}$. It is obvious that the lack of rewriting of these two attributes is due to either the lack of corresponding attributes in the schema of LuDB, or the lack of mappings between StuartDB and LuDB that encapsulate the correspondence of these attributes. Nevertheless, the rewritten version never has additional 'select' attributes, compared with the original one.

***Observation:*** *It is clear that 'select' attributes missing in the rewritten query version influence negatively the overall similarity of the queries.*

**(2) 'where' conditions:**

Yet, things are more vague with the query conditions. There are several situations and we consider each separately.

**(2a) additional value constraints:** In our example $Q_{LuDB\_sr}$ has the additional condition $P.Age < 13$. Is this condition an additional constraint to the query compared with the non-conditional $Q_{StuartDB\_sr}$? In order to find out, we have to consider the circumstances under which the mapping that contributed this condition, $M_{StuartDB\_LuDB}$, was created. In our case, Dr Lu is a pediatrician, and, thus, he stores in his database information about kids, i.e. P.Age $< 13$ for all data in LuDB. Therefore, the corresponding additional condition in $Q_{LuDB\_sr}$ is not actually an additional constraint, since the set of returned tuples is the same if the posed query includes or not this condition. However, if Dr Lu is a family doctor, but for some reason his database maintains the mapping $M_{StuartDB\_LuDB}$ with StuartDB, the condition on $P.Age$ is an actual additional constraint, since $Q_{LuDB\_sr}$ returns fewer tuples than if the condition is eliminated. In general, additional value conditions in rewritten queries either restrict or do not influence the result of the query. This depends on the reasoning that created the mappings used for the query rewriting.

***Observation:****Since we are not able to know the logic beneath mapping creation, we consider additional value conditions as restrictive, and, therefore, that they decrease the similarity of the queries.*

**(2b) additional joins on non-key attributes:** Beyond value conditions, SPJ queries have joins either on relation keys or just plain attributes. As far as additional joins on plain attributes are concerned, we can follow the same rationale as in the previous paragraph. We can conclude that, in the same way as with additional value conditions, additional joins on plain attributes can be considered as more restrictive, in the general case.

***Observation:****Additional joins on non-key attributes decrease query similarity.*

**(2c) additional joins on key attributes:** However, is this the case for joins on relation keys? Consider again the successive rewritings $Q_{StuartDB\_sr}$ and $Q_{LuDB\_sr}$ of the motivating example. The second query has a join on relation keys, $D.Did = P.Did$, whereas the first does not have any. Easily, we can see that this join is necessary in order to 'select' both attributes $P.Insurance\sharp$ and $D.AvgFever$; thus, it does not restrict the query answer.

Yet, a minor objection to this reasoning is that two relations joined by their keys do not coincide with one relation that contains all their attributes: the first contains only the tuples of the two joined relations that have common key values, whereas the second can contain even the tuples of the two relations that have non-matching key values [4].

Furthermore, a join on keys restricts the query answer, if the rewritten version does not contain 'select' attributes from both parts of the join. For example, suppose that $Q_{StuartDB\_sr}$ does not contain in the 'select' clause the attribute $P.Insurance\sharp$. However, the only way to rewrite it is through the available mapping $M_{StuartDB\_LuDB}$. Hence, the rewritten version on LuDB will be:
$Q'_{LuDB\_sr}$:

```
SELECT  D.AvgFever
FROM    Disease D, Patients P
WHERE   D.Did = P.Did, P.Age < 13
```

In this case, even though the additional join is on relation keys, it does not serve as an associative action and it does restrict the query answer: in the absence of it the query would return additionally the tuples of relation *Disease* that do not have a matching *Did* with a the *Pid* of a tuple in relation *Patients*.

***Observation:*** *We consider additional joins on relation keys as neutral to query similarity, if the existence of 'select' attributes is based on the existence of the join; otherwise, we consider additional joins on relation keys as a negative influence to query similarity.*

**(2d) absent joins on key attributes:** Finally, let us consider joins on relation keys that exist in the source query, but are absent in the rewritten version. For example, this is the case of $Q_{orig}$ and $Q_{StuartDB\_sr}$. The two joins on relation keys in the former are not present in the latter. Should this element absence in the rewritten version be considered as a reformulation failure? The answer is no, obviously: since the query rewriting is performed using the mapping $M_{StuartDB\_DavisDB}$ that combines the relations *Visits*, *Disease* and *Treatment* from DavisDB in a correspondence to the *Treatments* relation in StuartDB, the two joins of $Q_{orig}$ are "*consumed*", in a way, during the complete rewriting through the discussed mapping. Thus, the absence of the two joins in the rewritten version does not mean that the mechanism failed to rewrite them, but that they are encapsulated in the mappings used for the rewriting, and they are not needed in the new query.

But, what if the joins of $Q_{orig}$ were not present in the mapping $M_{StuartDB\_DavisDB}$? For example suppose that StuartDB uses for the rewriting $Q_{orig}$ the mappings:
$M1'_{StuartDB\_DavisDB}$:

[4] Assume that there are two relations $R1(x,y)$ and $R2(x,z)$ with the same key, $x$, and a relation that contains all attributes of $R1$, $R2$: $R(x,y,z)$ with key $x$. The tuples of each one of $R1/R2$ that cannot be joined with a tuple of $R2/R1$, would have a corresponding tuple in $R$, for which the attributes corresponding to $R2/R1$ would be null (outer join)

Treatment$_{StuartDB}$(Pid, _, _, Symptom, _, DisDescr):- Visits$_{DavisDB}$(Pid, _, Did), Disease$_{DavisDB}$(Did, DisDescr, Ache), $\{Symptom = Ache\}$
and
$M2'_{StuartDB\_DavisDB}$:
Treatment$_{StuartDB}$(_, _, _, _, TreatDescr, _):- Treatment$_{DavisDB}$(_, Drug, _), $\{TreatDescr = Drug\}$ [5],
The rewritten query on StuartDB would be:
$Q'_{StuartDB\_sr}$:

```
SELECT  T1.Pid, T1.DisDescr, T1.Symptom, T2.TreatDescr
FROM    Treatment T1, Treatment T2
```

The second join of $Q_{orig}$, 'D.Did = T.Did', is not encapsulated in the above mappings. It is obvious that the lack of this join in the mappings and the rewritten version results in a cartesian product in the relation StuartDB.Treatment; thus, the lack of the key join affects really badly the reformulation of $Q_{orig}$.

***Observation:*** *We consider that joins on relation keys are satisfied, i.e. explicitly or implicitly present in a query rewriting, if they are present either in the mappings used for the reformulation or in the rewritten version itself. If joins on keys are not satisfied, we consider that their lack in the rewritten version affects negatively the similarity with the source query.*

In addition to the above, any other missing constraints (i.e. value constraints or joins on non-key attributes) are considered to have a bad impact on query similarity.

**(3) corresponding 'select' attributes and 'where' conditions:**
Suppose that the only available mapping in order to rewrite $Q_{orig}$ is $M1'_{StuartDB\_DavisDB}$. Then the rewriting procedure would produce the query version:
$Q''_{StuartDB\_sr}$:

```
SELECT  T.Pid, T.DisDescr, T.Symptom
FROM    Treatment T
```

Again, neither the rewritten query nor the used mapping contain any form of the join 'D.Did = T.Did' of $Q_{orig}$. Moreover, in this case the 'select' attribute Treatment.Drug of $Q_{orig}$ is not mapped in StuartDB, and, thus, it is not present in $Q''_{StuartDB\_sr}$. So, the lack of both the aforementioned join and attribute should influence negatively the similarity of the source and the rewritten query. However, the attribute Drug is not really worth of rewriting, even if this is possible, (e.g. if the mapping $M2'_{StuartDB\_DavisDB}$ is available), if the join 'D.Did = T.Did' cannot be rewritten: as we have discussed previously, the result would be a cartesian product, which in general cannot be considered a good rewriting.

The question that arises from this situation is whether the lack of these two elements should affect the query similarity in a correlated or separate way. Again, the answer to

[5] The reader can observe that the mapping $M2'_{StuartDB\_DavisDB}$ does not map tuples 1-1; yet, this is a possible mapping and its meaning is: "tuples in DavisDB.Treatment correspond to tuples in StuartDB.Treatment where the respective attributes Drug and TreatDescr have the same value "

this question depends on how users think in order to form an originally posed query. If users use joins on relation keys only as an associative means for retrieving attributes from distinct relations, then the role of these joins is supportive to ’select’ attributes and the impotence of their reformulation leads to the impotence of retrieving the supported attributes from the target database; these joins cannot be present alone, i.e. without the attribute(s) they support. Therefore, as far as query similarity is concerned, the lack of such rewritten joins should be correlated with the lack of the supported rewritten attributes. Nevertheless, in case we assume that users create queries without any specific logic, key joins cannot be thought of as associative to retrieved attributes, in general. Hence, their lack should be considered as not affecting query similarity.

The second assumption is more conservative than the first, because it considers that two independent query features are missing from the rewritten version. Moreover, the second assumption does not affect the query rewriting procedure, since query features can be considered separately for rewriting (Section 4); yet, in agreement with the first assumption, combinations of ’select’ attributes / ’where’ conditions should be spotted.

***Observation:*** *We consider that the presence of each query feature is independent from the presence of the rest and they affect query similarity in a separate way*.

In the same spirit, we have to consider whether the retrieval of an attribute should be correlated with value conditions on the same attribute. For example, suppose a query similar to $Q_{LuDB\_sr}$, where Patient.Age is a ’select’ attribute:
$Q''_{LuDB\_sr}$:

```
SELECT  P.Insurance#, D.AvgFever, P.Age
FROM    Disease D, Patients P
WHERE   D.Did = P.Did, P.Age < 13
```

Should the lack of Patient.Age in a rewriting of $Q''_{LuDB\_sr}$ be correlated with the lack of the condition ’Patient.Age $<$ 13’ in the same rewriting? As in the case of joins on keys discussed previously, the answer to this question can be either ’yes’ or ’no’. On one hand, the presence of the value condition in the rewritten query depends on the presence of the respective ’select’ attribute, for if the second is not mapped in the target database, both of them cannot be rewritten. On the other hand, a user may create the original query with or without the value condition, meaning that in general the presence of the ’select’ attribute in the original query does not depend on the presence of the respective value condition or vice versa.

***Observation:*** *Following the second reasoning, we do not correlate ’select’ attributes with respective value conditions in the calculation of query similarity.*

## 3.2 Query Similarity Criteria

Based on the above observations, we want to form the criteria for the assessment of query similarity. The rough outcome of the earlier discussion is that missing or additional query features, i.e. ’select’ attributes or ’where’ conditions should be considered decreasingly in query similarity, from a conservative point of view. We choose a conservative point of view in order to determine a correct estimation of query similarity in any context of P2P database applications.

We have to refine our observations by ordering the importance of the role of the various missing or additional query features. First, we reckon that key attributes are highly important in a relational schema since their values uniquely prescribe the values of other attributes. We think that the role of keys in queries is as important as in the schema itself, no matter if such an attribute appears in a 'select' or 'where' clause. Thus, deficient rewritings of key attributes may result in severe semantic deviations from the original query. Second, 'select' attributes represent what information the user requires actually. Thus, their lack in the rewritten query is decisively irreparable. Third, even though the lack of join conditions is a negative factor for query similarity, it results in a query version that retrieves a superset of the data that would be retrieved by a query with the rewritten joins. Furthermore, the lack of value constraints has the same effect in the query as the lack of joins. However, the lack of joins probably results in much bigger supersets of retrieved data than the lack of value constraints. Finally, the introduction of new value constraints, joins on non-key attributes is considered a deficiency . Yet, new conditions produce a rewriting that is classically contained in the source query; thus, we think of the introduction of new conditions as the weakest criterion of all.

Thus, we form the following criteria for the definition of the similarity of two query versions. The criteria are ordered according to their importance in query similarity.

Cr1 Key attributes are rewritten, no matter what their position in the query is
Cr2 'select' attributes are rewritten
Cr3 Join attributes are rewritten
Cr4 Constrained attributes (beyond join ones) are rewritten
Cr5 There are no additional parts in the query: new value constraints and joins on non-key attributes [6]
- a There are no new value constraints
- b There are no new joins on non-key attributes
- c There are no new joins on key attributes that are necessary for the rewriting of 'select' attributes

Overall, the above ordering of the criteria is based on the rationale that the most important elements of a query are the attributes that are keys or 'select' attributes. Joins are very important; yet their lack results in supersets of answers that the peer might be able to refine. Finally, additional conditions (thus, classically contained rewritings) are considered the most lossless rewritings. The lack of rewriting of keys or query conditions (joins and value constraints) results in answers that are not sound whereas the lack of rewriting of 'select' attributes and additional conditions result in answers that are not complete. Since the importance of answer soundness over completeness or the opposite is application-dependent[7], we do not base the ordering of the criteria solely on them.

[6] We consider all additional parts in a query in this last criterion. We distinguish the parts according to their role in $(a)$, $(b)$ and $(c)$ as shown. All of them have roughly the same priority; yet, in a refined similarity metric, priority can be given according to lexicographical order. Finally, the criterion $5(c)$ may be eliminated in more conservative similarity metrics. A thorough investigation of the impact of these criteria to a variety of application fields is out of the scope of this work.

[7] For example in a medical application, as in the motivating example, soundness may be more important than completeness, whereas in an application of multimedia content the opposite may hold.

## 3.3 Discussion on Alternative Approaches to Query Similarity Metrics

The ordering of the criteria in Section 3.2 is formed on the atomic importance of each query element. In order to decide about the overall similarity of the original and the rewritten query version there is a need for a metric that correlates the importance of the separate query elements.

As we have seen in Section 3.1, the similarity guidelines along which we can form a query similarity function take into consideration the 'select' attributes $A$ and the 'where' conditions, i.e., the value constraints $C$ and the join conditions $J$, individually. The rewriting of query parts can be explicit in the rewritten query or implicit in the used mappings; for both cases we use the function symbol *rewr* to refer to the rewritings of query parts.

Beyond the separate rewriting of $A$, $J$ and $C$, the peer has to decide if there is any correlation among them. The peer's policy concerning this correlation depends on the importance it gives to the rewriting of $J$s in the translated query, as we have discussed earlier. Moreover, the correlation of the different query elements in the similarity function has a respective impact on their separate importance in the query. Thus, a strong correlation of $J$ elements with the rest or among themselves (e.g. the following functions (2) and (3)), achieves the domination of $J$ elements in total but also a more critical importance of each one of them, separately (especially function (3)). Beyond the general correlation of query elements, weights can be associated with each one of them so that tuning to the special needs of each query can be performed.

If the peer is, in a way, optimistic, then it can consider that all three sets $A$, $C$ and $J$ are of equal importance and the non-rewriting of any of them does not influence the rewriting of the rest. Such peers may be more interested in complete than sound answers. The function that quantifies this policy is the following:

$$\frac{\sum rewr(J) + \sum rewr(A) + \sum rewr(C)}{\sum J + \sum A + \sum C} \tag{1}$$

However, a reasonable policy is to consider the rewriting of $J$ to be more important than the satisfaction of $A$ and $C$. More specifically, a peer can decide that the rewriting of $A$ and $C$ depends on the average of the rewritten $J$:

$$\frac{\sum rewr(J) \cdot (\sum rewr(A) + \sum rewr(C))}{\sum J \cdot (\sum A + \sum C)} \tag{2}$$

Or the rewriting of $A$ and $C$ depends on the rewriting of all $J$:

$$\frac{\prod rewr(J) \cdot (\sum rewr(A) + \sum rewr(C))}{\prod J \cdot (\sum A + \sum C)} \tag{3}$$

Moreover, a peer can decide that the rewriting of $J$ influences only the $A$ and $C$ referring to the same relations. In this case the two above functions, (2) and (3) are altered respectively as follows:

$$\frac{\sum_R (\sum rewr(J_R) \cdot (\sum rewr(A_R) + \sum rewr(C_R)))}{\sum_R (\sum J_R \cdot (\sum A_R + \sum C_R))} \tag{4}$$

$$\frac{\sum_R(\prod rewr(J_R)\cdot(\sum rewr(A_R)+\sum rewr(C_R)))}{\sum_R(\prod J_R\cdot(\sum A_R+\sum C_R))} \tag{5}$$

where the index $R$ indicates the relations to which the attributes in $J_R$, $A_R$, and $C_R$ refer, respectively. Peers that use functions (2), (4) or (3), (5) may be more interested in sound than complete answers.

The definition of *rewr* can vary depending on what a peer considers that contributes to a 'good' or a 'bad' rewriting. Moreover, the result of *rewr* has to be a number that quantifies the quality of the rewriting. Depending on the implementation, the *rewr* function quantifies the 'good' and the 'bad' rewriting, either as a boolean function (i.e. *rewr* produces only two values, for example $\{0,1\}$) or as a function that produces a range of continuous or non-continuous values (for example *rewr* can produce a result in $[0,1]$). The peer's policy for partial rewriting of query parts can take into consideration possible differences of data types and ranges and, also, the difficulty to transform encodings. For example, in the motivating example we have assumed that the attribute *Drug* in DavisDB is rewritten as the attribute *TreatDescr*; however, the first might not be so accurate as the latter. Also, suppose that the original query contained the following condition: Pid $>$ 505. Even though the assumed mapping in the motivating example matches *Pid* to *Insurance#*, the translation and thus the satisfaction of this condition may not be possible: *Insurance#* $>$ '505' has most probably no meaning for Dr Lu's database. Situations like the one just described hint that careless translation of a $C$ not only may not actually satisfy the original $C$ but also deteriorate significantly the whole query: the condition *Insurance#* $>$ 505 could lead to a translated query that gives an empty answer. Thus, we do require a safe and conservative translation of $C$s. In this spirit, the condition *Pid* $>$ 505 would not be rewritten in the new query, since a general rule could prohibit the transformation of numerical values to strings. Nevertheless, data mapping is out of the scope of this work.

Nevertheless, a thorough investigation of the applicability of the various correlations among the rewritings of query parts as well as of the various implementations of *rewr* is out of the scope of this work. In the following we present the approach we have taken in *GrouPeer* for the construction of a moderate and generally applicable query similarity metric.

## 3.4 Our Approach to a Query Similarity Metric

In our approach a query is considered to be a set of elements, one for each 'select' attribute and one for each 'where' condition. A propagated query version, $Q'_{sr}$, that is derived through rewritings on an initial query, $Q_{orig}$, maintains all the elements of the latter. Thus, $Q'_{sr}$ may contain elements that are not present in its SPJ form. These elements are marked as non-present. The absence of 'select' attributes means that $Q'_{sr}$ is degraded by them w.r.t $Q_{orig}$. However, this is not the case with 'where' clause conditions: the absence of a condition is a result of either the presence of it in a mapping used to produce the $Q'_{sr}$, or the inadequacy of mappings to rewrite the condition. In the first case, even though the condition is not present structurally in the SPJ form of $Q'_{sr}$, it is semantically encapsulated. In the second case, $Q'_{sr}$ is degraded by the lack of this condition.

**Definition 1.** *A query Q propagated in the P2P database system is a pair of two sets:* $Q = \{Sl, Cn\}$*; Sl is a set of elements* $Sl = \{E_1, .., E_m\}$*; each element* $E_i, i = 1, .., m$ *corresponds to a 'select' attribute* $A_i \equiv R.A$ *and an indication, 'pres', whether* $A_i$ *is present in Q or not. Thus,* $Sl.E_i = \{A_i, pres\}$*,* $pres \in \{true, false\}$*. Cn is a set of elements* $Cn = \{E_1, .., E_n\}$*; each* $E_i$ *corresponds to a 'where' clause condition* $C_i$ *with an indication 'pres', whether* $C_i$ *is present in the query or not and an indication 'st' that shows if the condition is satisfied. Thus,* $Cn.E_i = \{C_i, pres, st\}$*,* $pres, st \in \{true, false\}$ *and* $C_i \equiv R.A = R'.A / R.A\ \varphi\ const$*, where* $\varphi \in \{=, >, <, \}$*, const is a data value, for* $i = 1, .., n$ *and R.A, (R'.A) denotes the attribute A of relation R, (R').*

A query condition is *satisfied* if it is explicitly or implicitly (i.e. through the employed mappings) present in a query rewriting. For clarity, we denote the set-of-elements form of a query $Q$ as $Q^{soe}$.

**Example** The set-of-elements form of $Q_{orig}$ is:
$Q^{soe}_{orig} = \{\{\{Pid, true\}, \{DisDescr, true\}, \{Ache, true\}, \{Drug, true\}, \{Dosology, true\}\}, \{\{\{Visits.Did = Disease.Did, true, true\}, \{Disease.Did = Treatment.Did, true, true\}\}\}$.
We omit the relation names wherever this is possible, in order to save space.

It is important to notice that the similarity of two queries is confined by the semantic similarity of their elements. Hence, the similarity measure we are seeking should be in the same spirit as such measures in the field of schema matching (e.g. [28]) and matching taxonomies (e.g. [11]). Specifically, if each element of $Q_{orig}$ is semantically matched totally with the respective element of $Q_{rewr}$ and $Q_{rewr}$ does not include elements that cannot be matched with elements of $Q_{orig}$ (i.e. $Q_{rewr}$ does not have more constraints than $Q_{orig}$), then $Q_{orig}$ is semantically identical with $Q_{rewr}$.

Thus, we introduce the function *sat* that takes as input two query elements $E_i$ and $E'_i$ and returns the set of concepts of $E_i$ that could *not* be matched with concepts of $E'_i$ [8]. The following is the formal definition of a concept:

**Definition 2.** *Considering a relational schema S, a distinct concept corresponds to each R.A where A is an attribute of relation* $R \in S$*.*

Note that comparing $E_i$ and $E'_i$ is safe, meaning that the index $i$ indicates that both elements refer to the same query element ('select' attributes, $E_i \in Sl$, or 'where' conditions, $E_i \in Cn$).

Query elements $E_i$ can be implicit or explicit in the regular SQL form: explicit are those that appear in the SQL form and implicit are those that are encapsulated in mappings used for reformulation. The function *sat* compares elements either of their explicit or implicit existence in the SQL form. It is straightforward that for two queries $Q_{orig} = \{Sl, Cn\}$ and $Q_{rewr} = \{Sl', Cn'\}$, the function $sat(Sl.E_i, Sl'.E'_i)$ exports the concepts of $E_i$ for which $E'_i.pres = false$ and the function $sat(Cn.E_i, Cn'.E'_i)$ exports the concepts of $E_i$ for which $E'_i.st = false$ (i.e. $E'_i.pres = false$, too). Also, note that *sat* is not reflective: it exports the concepts of $E_i$ not matched in $E'_i$ but not the opposite.

[8] We do not assume that peers share an ontology. Yet, each peer may or may not possess and use dictionaries or ontologies in order to perform matching. If peers do not use any kind of dictionary or ontology, then concept matching is as simple as keyword matching.

Also, it is obvious that for $E'_j$ for which there is no corresponding $E_j$, *sat* compares $E'_j$ with the $\emptyset$ and exports all the concepts involved in $E'_j$. The *sat* function calculates a set of concepts that represents the semantic dissimilarity of two query elements. Thus, $|sat(E_i, E'_i)|$, is a quantification of the dissimilarity of the pair of $E_i$, $E'_i$. Based on *sat* values of all the elements involved in $Q_{orig}$ and $Q_{rewr}$, $M_{sim}$ can calculate the overall semantic similarity of the two query versions. The following is the formal definition of the default similarity metric that *GrouPeer* employs.

**Definition 3.** *For two query versions $Q_{orig} = \{Sl, Cn\}$, $Q_{rewr} = \{Sl', Cn'\}$ and a set of user-specified weights $w_{Q_{orig}}$ that denote the importance of each element in the semantics of $Q_{orig}$:*

$$M_{sim}(Q_{orig}, Q_{rewr}) = 1 - \frac{\sum_i w_i \cdot |sat(E_i, E'_i)| + \sum_j w_j \cdot |sat(E'_j, \emptyset)|}{\sum_i w_i \cdot |sat(E_i, \emptyset, )|} \tag{6}$$

*where $E_i$'s are elements of $Q_{orig}$ and $E'_i$'s,$E'_j$'s are elements of $Q_{rewr}$ and $\sum w_i = 1$. Specifically, if $E_i \in Sl$ then $E'_i \in Sl'$ and if $E_i \in Cn$ then $E'_i \in Cn'$. Also, $E'_j \in Cn'$. Moreover, by default the weights $w_i$ have values along the lines of the similarity criteria of section 3.2. The $w_j$ weights correspond to the criterion Cr5 about additional conditions.*

The metric $M_{sim}$ is structured such that dissimilar elements diminish its value and perfect similarity is represented by $M_{sim} = 1$. The proposed function $M_{sim}$ is constructed along the lines of the associative function (1); thus, we adopt a conservative view of query similarity that is more generic and more suitable to test in various applications. [9].

*GrouPeer* does not predefine a similarity threshold below which a query should not be rewritten. Instead, we recognize that the need (and quality standards) of each user about peer information is unique. Therefore, the similarity threshold according to which a query rewriting should be accepted or disregarded should be tuned by the user. Actually, the user has the power to tune the value of weights for all the query elements, and, thus decide if the lack of the rewriting of each such element is acceptable or not by her standards on the quality of answers that she expects from the system.

The above proposition about the form of the $M_{sim}$ function is not restrictive: as mentioned before, semantic similarity is a subjective issue, therefore, its calculation formula can differ from one peer to the next. Beyond this, defining semantic similarity for a wide range of queries and contexts is a complicated matter, since it not only depends on concept correspondences, but also on interrelations of concepts (ontologies).

## 4 Query Reformulation

We assume that peers own a query reformulation mechanism based on existing query rewriting algorithms, yet it enables the production of a rewritten version $Q_{rewr}$ from the original query $Q_{orig}$, where:

[9] Moreover, in cases of query conditions with arithmetic comparisons, the *sat* function can be implemented so that it takes care of rewritten versions that have narrower constraints than the original version (for example X $<$ 10 can be rewritten to X $<$8 [3])

– $Q_{rewr}$ maintains in the 'select' clause all the 'select' attributes of $Q_{orig}$ that can be rewritten through the available mappings. Thus, in conjunctive form, the head of $Q_{rewr}$ is a projection of the head of $Q_{orig}$.
– all the 'where' conditions of $Q_{orig}$ that cannot be rewritten through the available mappings are ignored. The rest are rewritten and included in $Q_{rewr}$. In conjunctive form, $Q_{orig}$ and $Q_{rewr}$ have mappings between attributes of predicates even if predicates themselves cannot be mapped.

As an example of the first rule above, remember the original query posed by Dr Davis, $Q_{orig}$; it is not possible to rewrite it to the schema of StuartDB, because the 'select' attribute Treatment.Dosology of $Q_{orig}$ is not mapped in StuartDB. Thus, this attribute is ignored in the classical query rewriting procedure.

As an example for the second rule above, consider the original query, $Q_{orig}$, augmented with a value condition:
$Q_{orig\_changed}$:

```
SELECT  V.Pid, D.DisDescr, D.Ache, T.Drug, T.Dosology
FROM    Disease D, Treatment T, Visits V
WHERE   V.Did = D.Did AND D.Did = T.Did AND T.Dosology = 5mg
```

Since the attribute Treatment.Dosology is not mapped in StuartDB, there cannot be a classically contained rewriting of $Q_{orig\_changed}$, because the condition 'T.Dosology = 5mg' cannot be rewritten. Thus, this condition is ignored.

In this work, we use the terms *reformulation* and *rewriting* interchangeably, meaning the query translation according to the principles described above. In order to achieve the above reformulation, the algorithm has two conceptual steps:

1. Pre-processes the incoming query $Q_{inc}$ and produces the version $Q_{inc\_preprocessed}$ that contains the part of $Q_{inc}$ (i.e. the 'select' attributes and 'where' conditions) that can be altogether reformulated by a single mapping.
2. Then, $Q_{inc\_preprocessed}$ is rewritten with the classical query rewriting algorithms.

These two steps are conceptually distinct. However, depending on the implementation, they can be performed either in a sequential way or in a way that is partially sequential and partially parallel, i.e. some parts of the preprocessing procedure can be merged with the query rewriting itself [10].

**Example** Assume that StuartDB and LuDB have the following mapping:
$M1_{StuartDB\_LuDB}$:
Treatment$_{StuartDB}$(Pid, _, _, _, _, _):- Disease$_{LuDB}$(Did, _, _),
Patients$_{LuDB}$(Insurance♯, Did, _, _), Age $< 13$, $\{Pid = Insurance\sharp\}$

Then the query $Q_{StuartDB\_sr}$ is preprocessed so that the 'select' attribute 'Treatment. Symptom' is eliminated. The preprocessed query version is:
$Q_{StuartDB\_sr\_preprocessed}$:

[10] For example if query rewriting using LAV mapping is performed with the bucket algorithm [25] then preprocessing has to be done before query rewriting; yet, if query rewriting is performed with the minicon algorithm [35] the preprocessing can be integrated in the algorithm itself.

```
SELECT  T.Pid, T.DisDescr, T.TreatDescr
FROM    Treatment T
```

Now, $Q_{StuartDB\_sr\_preprocessed}$ can be rewritten in the classical way using $M1_{StuartDB\_LuDB}$.

**Query Preprocessing Guidelines**

In order to pre-process a query and produce the input for the rewriting algorithm, we choose the mapping(s) with respect to which we will perform the preprocessing. As aforementioned, we want to choose the mappings that *best* rewrite the query. The best rewritten version is valuated with respect to the similarity criteria defined in the previous section. The mappings used for the rewriting are actually the exclusive means that provide the rewritten query features thus, the structure of the mappings reflects the rewritten query. Also, mappings are actually queries (or pairs of queries) themselves. Thereupon, we base our decision for the selection of mappings on the query similarity criteria defined in Section 3.

There is a variety of query-mapping combinations that we can come across during the mappings selection procedure. We discuss the combinations of queries and considered mappings that do not match completely. The following is a categorization of the main query-mapping combinations that we can come across, where attributes of relations involved in the query are missing from the mapping. This categorization is complete and other combinations actually fall into one or more of these categories. The categorization follows the lines of query similarity aspects.

**Case A:** Considering GAV mappings

In this case we consider attributes of relations that are missing from the mapping. These relation attributes may appear in the query ('select' or 'where' clause), or not. Attributes that appear in the query but not in the mapping are related to one of the criteria defined in Section 3.2. Moreover, in this case we consider additional conditions on mapped attributes that appear in both the query and the mapping.

1. For attributes that are not present in the 'select' clause:
   (a) The query is $Q(x,y) : -P(x,y,z)$ and the mapping is $P(x,y,\_) : -P'(x,y)$
   (b) The query is $Q(x,y) : -P(x,y,z), z =' c'$ and the mapping is $P(x,y,\_) : -P'(x,y)$
   (c) The query is $Q(x,y) : -P(x,y,z)R(z,w)$ and the mapping is $R(\_,w) : -R'(w)$
   (d) The query is $Q(x,y) : -P(x,y,z)R(z,w)$ and the mapping is $R(z,\_) : -R'(z)$
   (e) The query is $Q(x,y) : -P(x,y,z)$, the mapping is $P(x,y,z) : -P'(x,y,z), z =' c'$
   (f) The query is $Q(x,y) : -P(x,y,z)$ and the mapping is $P(x,y,z) : -P'(x,y,z)R'(z)$
2. For attributes that are present in the 'select' clause:
   (a) The query is $Q(x,y) : -P(x,y,z)$ and the mapping is $P(x,\_,z) : -P'(x,z)$
   (b) The query is $Q(x,y) : -P(x,y,z)$, the mapping is $P(x,y,z) : -P'(x,y,z), x =' c'$
   (c) The query is $Q(x,y) : -P(x,y,z)$ and the mapping is $P(x,y,z) : -P'(x,y,z)R'(x)$

Cases 1(b) and 1(c) denote missing value constraints (criterion Cr4) and joins (criterion Cr3), respectively. Cases 1(a,d) denote missing attributes from relations of the query;

yet these attributes do not appear in the query itself [11]. Cases 1(e,f), 2(b,c) denote additional ’where’ conditions (criterion Cr5). Case 2(a) denotes a missing ’select’ attribute (criterion Cr2).

**Case B:** Considering LAV mappings

In this case we consider attributes of relations that are involved in the query but are missing from the mapping.

1. ’select’ attributes of the query are missing from the mapping.
   (a) The query is $Q(x,y) : -P(x,y,z)$ and the mapping is $Q'(x) : -P(x,y,z)$ or $Q'(x,z) : -P(x,y,z)$
2. ’where’conditions of the query are not mapped through the mapping:
   (a) The query is $Q(x,y) : -P(x,y,z), z =' c'$ and the mapping is $Q'(x,y) : -P(x,y,z)$
   (b) The query is $Q(x,y) : -P(x,y,z)R(z)$ and the considered mapping is $Q'(x,y) : -P(x,y,z)$ and there is no mapping for $R(z)$ [12]
3. additional ’where’ conditions
   (a) The query is $Q(x,y) : -P(x,y,z)$ and the mapping is $Q'(x,y) : -P(x,y,z), z =' c'$
   (b) The query is $Q(x,y) : -P(x,y,z)$ and the mapping is $Q'(x,y) : -P(x,y,z)R(x)$
   (c) The query is $Q(x,y) : -P(x,y,z)$ and the mapping is $Q'(x,y) : -P(x,y,z)R(z)$

Case (1a) corresponds to criterion Cr2. Case 2(a) denotes a missing value constraint (criterion Cr4), whereas case 2(b) denotes a missing join (criterion Cr3). Note that both cases can refer to mappings with additional ’where’ conditions (criterion Cr5). Case 3(a) denotes an additional value constraint and cases 3(b,c) denote additional joins on relation attributes that appear or not in the query (criterion Cr5).

**Case C:** Considering GLAV mappings[13]

1. ’select’ attributes of the query are missing from the mapping.
   (a) The query is $Q(x,y,z)$ and the mapping is $Q(x,y,z) : -Q'(x,y)$
2. The mapped query has additional ’select’ attributes:
   (a) The query is $Q(x,y)$ and the mapping is $Q(x,y) : -Q'(x,y,z)$

We select mappings according to the criteria of Section 3.2. The lack of attributes of relations involved in the query that do not appear either in the ’select’ or the ’where’ clause, such as in Case A 1(a,d), is not taken into consideration: the mapping correspondences denote that the predicates match, even if not all their attributes match.

[11] These cases do not correspond to any criterion of section 3.2: in section 3 we have focused on similarity of queries and we have not considered similarities of attributes that are not involved in queries. Nevertheless, we present cases 1(a,d) for completeness

[12] Actually, classical LAV rewriting algorithms (e.g. [35]) cannot rewrite $R(z)$ even if there is a mapping for it, since the join on $z$ cannot be rewritten.

[13] We only consider the head of queries in GLAV mappings. Refining our categorization for the bodies of these queries is future work

In order to impose the criteria of section 3.2 we construct an associative similarity function. Specifically, for a query subgoal $g$ [14] and a mapping $M$, the associative function $M_{as}$ quantifies the lack of non-matched attributes.

$$M_{as}(g,M) = \sum_{i=1}^{|g|} w_i \cdot a_i, w_i \in \{w_k, w_s, w_j, w_c\}, a_i \in \{0,1\} \tag{7}$$

where $|g|$ denotes the arity of $g$, i.e. the total number of attributes in $g$ involved in the query. Also, each subgoal attribute, $i$, has a weight, $w_i$ that denotes if it is a key, a 'select', a join or a constrained attribute. If it has more than two such characteristics, it keeps the one that is higher in the hierarchy of the criteria. Accordingly, the weights for a key, a 'select', a join or a constrained attribute is $w_k$, $w_s$, $w_j$ and $w_c$, respectively. We require that $w_k > w_s > w_j > w_c$. Finally, $a_i$ denotes if the respective attribute is matched in the mapping ($a_i = 0$) or not ($a_i = 1$).

We use the following form that estimates the matching ability of the selected mappings:

$$Sim_g(Q,\mathcal{M}) = 1 - \frac{\sum_j Sim_{g_j}.M_{as} + \sum_k w_a}{\sum_j \sum_{i=1}^{|g_j|} w_{ji}}, w_i \in \{w_k, w_s, w_j, w_c\}, a_i \in \{0,1\} \tag{8}$$

$\mathcal{M}$ is the set of selected mappings and $Sim_{g_j}.M_{as}$ is the $M_{as}$ value for the $g_j$ subgoal; the weight $w_{ji}$ refers to the $i^{th}$ attribute of the $j^{th}$ subgoal. The weight for an additional condition is $w_a$ ($w_c > w_a$) and $\sum w_a$ represents the total weight of all the additional conditions in the selected mappings. Actually, the $Sim_g$ function speculates the result of $M_{sim}$ of section 3.4, by estimating if each query element can be rewritten with the considered mapping; $Sim_g$ breaks down query elements to attributes of query subgoals in order to facilitate the straightforward use of mappings in conjunctive form in the preprocessing algorithm. In this way, we manage to estimate if a mapping can be used for rewriting a query, without running the rewriting algorithm.

The algorithm that chooses which mappings will be used in the query rewriting procedure is shown in Figure 4. Briefly, the algorithm considers three sets of mappings: GLAV, GAV and LAV. It produces one subset of each set. Each such subset contains the mappings that are most similar to the corresponding query subgoals according to the aforementioned criteria of section 3.2. Finally, the algorithm chooses the subset of the three which has the highest $Sim_g$ value for the query.

In detail, the algorithm works as follows. In case of GLAV mappings we check the overall real similarity of the of the original and the rewritten version. In the case of GAV mappings we have to check which query elements can be satisfied by which GAV mappings; for query elements that can be rewritten with more than one mapping we have to check which one of them poses the fewest additional restrictions. The execution of this preprocessing is simple and straightforward. Similar actions have to be done for LAV mappings. In this case we have also to check if a mapping that satisfies a part of a query (a subgoal) is satisfied by the head or the body of the mapping. In the first

[14] SPJ queries can be considered in conjunctive form, where each conjunct is called a 'query subgoal' [43]

case we proceed as in the case of GAV mappings. Also, for query elements that are satisfied in the body of the mapping that is currently examined we check if these are value-constrained or if they are joined with subgoals that have not been examined yet. In these cases we also proceed as in the case of GAV mappings. However, for query elements that are joined with subgoals that have been examined, we have to check the mapping that has been selected and stored by the algorithm for these subgoals. If this is the same mapping as the one that is currently examined, then we proceed as in the case of GAV mappings. Else, this query element cannot be satisfied by the mapping and we reduce the value of the similarity of the currently examined mapping and subgoal accordingly.

## 5 Clustering of Peers

Recall that the goal of clustering is to enable peers to discover other peers with similar interests and schemas, that cannot be ’tracked’ otherwise, because they are hidden in query propagation paths by other peers with dissimilar interests.

Pairs of remote peers that exchange queries and answers learn gradually about the schema of each other; learning is performed through making queries an evaluating their answers, and is formed in mappings between the schemas of the two peers. Actually, each peer that receives a query tries to retrieve any query attributes that are lost in the successive rewriting of the query along the propagation path. The mappings formed between two remote peers encapsulate the common interest of the two peers, since they refer to the (necessary) schema parts on which they express and answer queries. If the peers decide to become acquainted, these mappings are already a language for their communication and alleviate the administrator’s load for creation of mappings for the new acquaintance. The peers accumulate the answers they receive from each peer; each time they receive a new answer from a specific peer, they compute the current overall respective similarity of answers by this peer. If this similarity exceeds a predefined user-tuned threshold, the peer may decide to ask this peer to be its acquaintee. The mappings that are formed during the query position-answering between them are used as an initial set of communication mappings.

Overall, the proposed methodology of making new acquaintances in the overlay leads to the restructuring and, moreover, the gradual clustering of the P2P system in groups with common interests. Thus, we refer to it as the ‘clustering process’. In what follows we describe the details of the clustering process and we give the respective algorithm. Also, we discuss the role of automatic schema matching and the protocol internals of *GrouPeer*.

### 5.1 Schema Mappings and Query Rewriting

The definitions in section 3.4 together with the following ones formalize the basic notions of a concept, a query and a mapping in forms suitable for the clustering process we describe next.

**Definition 4.** *Considering a source schema S and a target schema $S'$, a GAV/LAV/GLAV mapping between them $M(S,S')$ is the set $\{Cr_M(S,S'), Cond_M(S,S')\}$, where the set*

```
Input: the query to be rewritten $Q$
Output: A mapping $M$ or a set of mappings $\mathcal{M}$

– Step 1 Consider GLAV mappings:
    • Make a set $Sim_{GLAV} = \{M, M_{sim}\}$ and initiate $M = null^{a}$ and $M_{sim} = -\infty$
    • For each mapping $M$ represented as $Q' : -Q''$ compute $M_{sim}(Q, Q'') = M_{sim}(Q, Q') \cdot M_{sim}(Q', Q'')$*; if $Sim_{GLAV}.M_{sim} < M_{sim}(Q, Q'')$ replace $Sim_{GLAV}.M$ with $M$ and $Sim_{GLAV}.M_{sim}$ with $M_{sim}(Q', Q'')$
– Step 2 Consider GAV mappings:
  For each subgoal $g_j$ of $Q$:
    • Make a set $Sim_{g_j} = \{M, M_{as}\}$ and initiate $M = null$ and $M_{as} = -\infty$
    • For each mapping $M$ that matches the predicate of $g_j$ compute $M_{as}(g_j, M)$; if $Sim_{g_j}.M_{as} < M_{as}(g_j, M)$ replace $Sim_{g_j}.M$ with $M$ and $Sim_{g_j}.M_{as}$ with $M_{as}(g_j, M)$
  Make the set $Sim_{GAV} = \{\mathcal{M}_{\mathcal{GAV}}, Sim_g\}$, where $Sim_g(Q, \mathcal{M}) = 1 - \frac{\sum_j Sim_{g_j}.M_{as} + \sum_k w_a}{\sum_j \sum_{i=1}^{|g_j|} w_{ji}}$
– Step 3 Consider LAV mappings:
  For each subgoal $g_j$ of $Q$:
    • Make a set $Sim_{g_j} = \{M, M_{as}\}$ and initiate $M = null$ and $M_{as} = -\infty$
    • For each mapping $M$ that matches the predicate of $g_j$ compute $M_{as}(g_j, M)$;
    • in the following cases:
        ∗ if all the query attributes of $g_j$ are in the head of $M$
        ∗ if a query attribute $t$ of $g_j$ is value constrained and this constraint is included in the body of $M$
        ∗ if a query attribute $t$ of $g_j$ is joined with a subgoal $g_k$ that has not been examined yet.
      do: if $Sim_{g_j}.M_{as} < M_{as}(g_j, M)$ replace $Sim_{g_j}.M$ with $M$ and $Sim_{g_j}.M_{as}$ with $M_{as}(g_j, M)$
    • else (a query attribute $t$ of $g_j$ is joined with an examined subgoal $g_k$):
        ∗ if $Sim_{g_k}.M \neq M$ then do $M_{as}(g_j, M) = M_{as}(g_j, M) - w_t$
        ∗ if $Sim_{g_j}.M_{as} < M_{as}(g_j, M)$ replace $Sim_{g_j}.M$ with $M$ and $Sim_{g_j}.M_{as}$ with $M_{as}(g_j, M)$
  Make the set $Sim_{LAV} = \{\mathcal{M}_{\mathcal{LAV}}, Sim_g\}$, where $Sim_g(Q, \mathcal{M}) = 1 - \frac{\sum_j Sim_{g_j}.M_{as} + \sum_k w_a}{\sum_j \sum_{i=1}^{|g_j|} w_{ji}}$
– Step 4 Compare $Sim_{GLAV}.M_{sim}$, $Sim_{GAV}.Sim_g$ and $Sim_{LAV}.Sim_g$; depending on the highest value of the three, replace $\mathcal{M}$ with one of the sets $Sim_{GLAV}.M$, $Sim_{GAV}.\mathcal{M}$, $Sim_{LAV}.\mathcal{M}$

*$M_{sim}(Q, Q')$ is a function that quantifies the semantic similarity of two queries $Q$ and $Q'$ based on the proposed criteria.
```

[a] *null* denotes that there is no mapping

**Fig. 4.** Algorithm for the selection of mappings for the query rewriting

*of concept correspondences* $Cr_M(S,S') = \{R.A = R'.A' | R.A \in S, R'.A' \in S'\}$ *holds under the set of conditions* $Cond_M(S,S') = \{R_1.A = R_2.B$ *or* $R_1.A = const | R_1, R_2 \in S$ *or* $R_1, R_2 \in S'\}$*; const is a data value.*

Obviously, for each pair of concepts $\{R.A, R'.A'\}$ that each belong to a different schema, $R.A \in S$ and $R'.A' \in S'$, and that are corresponded through a mapping $M(S,S')$, there is one such pair in $Cr_M(S,S')$. A set of mappings between $S,S'$ is denoted as $\mathcal{M}(S,S')$.

For example, recall the mapping $M_{StuartDB\_DavisDB}$ from Section 1. This mapping is a set $\{Cr_M(StuartDB, DavisDB), Cond_M(StuartDB, DavisDB)\}$, where the set of concept correspondences is: $Cr_M(StuartDB, DavisDB) = \{Pid = Pid, Symptom = Ache, TreatDescr = Drug, DisDescr = DisDescr\}$, where for convenience we have eliminated the information about relations. Since the previous correspondences hold without value conditions: $Cond_M(StuartDB, DavisDB)\} = \{\{Visits.Did = Disease.Did\}, \{Disease.Did = Treatment.Did\}\}$.

Remember the mapping $M_{StuartDB\_LuDB}$. In the proposed form, the set of correspondences is $\{Cr_M(StuartDB, LuDB)\} = \{Pid = Insurance\sharp, Symptom = AvgFever\}$ and the set of conditions is $Cond_M(StuartDB, DavisDB)\} = \{Disease.Did = Treatment.Did\}, \{Age < 13\}$.

**Definition 5.** *For each correspondence* $R.A = R'.A' \in Cr_M(S,S') \in M(S,S')$*, the concepts,* $R.A$*,* $R'.A'$ *are considered equivalent.*

**Example** Assume that StuartDB poses the following query:
$Q1_{StuartDB}$[15]:

```
SELECT  T.Pid, T.DisDescr, T.Symptom, T.TreatDescr
FROM    Treatment T
WHERE   T.Date > 01/01/2000
```

The representation of this query in the proposed form is:
$Q1^{soe}_{StuartDB} = \{\{\{Pid, true\}, \{DisDescr, true\}, \{Symptom, true\}, \{TreatDescr, true\}, \{Date > 01/01/2000\}\}$. For simplicity we omit the relation names.

We assume that StuartDB and LuDB maintain the mapping $M_{StuartDB\_LuDB}$ (see Section 1). The rewriting of $Q1_{StuartDB}$ on LuDB is the following:
$Q1_{LuDB\_sr}$:

```
SELECT  P.Insurance#, D.AvgFever
FROM    Disease D, Patients P
WHERE   D.Did = P.Did, P.Age < 13
```

It is obvious that $Q1_{LuDB\_sr}$ coincides with $Q_{LuDB\_sr}$ (see Section 1). The set-of-elements form of $Q1_{LuDB\_sr}$ is:
$Q1^{soe}_{LuDB\_sr} = \{\{\{Insurance\sharp, true\}, \{DisDescr, false\}, \{AvgFever,$

[15] The condition of the query denotes that records with date after the date 01/01/2000 are requested. For convenience we use the symbol ’>’ to denote this request in the query and the following mapping.

$true\}$, $\{TreatDescr, false\}$, $\{\{Date > 01/01/2000, false, false\}, \{Disease.Did = Patients.Did, true, true\}, \{Age > 13, true, true\}\}\}$. This form denotes that the condition 'T.Date > 01/01/2000' of the initial query is neither present nor satisfied in the rewritten version on LuDB. Also, there is one additional condition, 'P.Age < 13', which constraints more the initial query. Note that the order of the elements in the $Q1^{soe}_{LuDB\_sr}.Sl$ set is the same as in the $Q1^{soe}_{StuartDB}.Sl$; thus, $\{Insurance\sharp, true\}$ corresponds to $\{Pid, true\}$, and so on. It is obvious that two of the 'select' elements of $Q1_{StuartDB}$ are missing from $Q1_{LuDB\_sr}$. $Q1_{LuDB\_sr}$ is degraded by the 'false' elements, the 'false' condition and the new condition.

Now suppose that the mapping between StuartDB and LuDB is:
$M2_{StuartDB\_LuDB}$:
Treatment$_{StuartDB}$(Pid, _, _, Symptom, _, _), Date > 01/01/2000:- Disease$_{LuDB}$(Did, AvgFever, _), Patients$_{LuDB}$(Insurance♯, Did, _, _), Age < 13, $\{Pid = Insurance\sharp, Symptom = AvgFever\}$

The rewritten version of $Q1_{StuartDB}$ on LuDB through $M2_{StuartDB\_LuDB}$ is again $Q1_{LuDB\_sr}$. However, in this case the set-of-elements form of $Q1_{LuDB\_sr}$ is:
$Q1^{soe}_{LuDB\_sr} = \{\{\{Insurance\sharp, true\}, \{DisDescr, false\}, \{AvgFever, true\}, \{TreatDescr, false\}, \{\{Date > 01/01/2000, false, true\}, \{Disease.Did = Patients.Did, true, true\}, \{Age > 13, true, true\}\}\}$. This denotes that the condition 'Date > 01/01/2000' is not present in the rewritten version but it is satisfied (through the mapping $M2_{StuartDB\_LuDB}$).

**Example** Furthermore, the set-of-elements form of $Q_{LuDB\_sr}$ is:
$Q^{soe}_{LuDB\_sr} = \{\{\{Insurance\sharp, true\}, \{DisDescr, false\}, \{AvgFever, true\}, \{Drug, false\}, \{Dosology, false\}\}, \{Disease.Did = Patients.Did, true, true\}, \{Age < 13, true, true\}\}\}$.

Note that, even though the SPJ form of $Q_{LuDB\_sr}$ and $Q1_{LuDB\_sr}$ is the same, their set-of-elements forms differs because they are derived from different initially posed queries.

Essentially, $Q'_{sr}$ keeps from $Q$ only the concepts and concept constraints that are present in the mappings. Also, $Q'_{sr}$ has more concept constraints than $Q$. We say that $Q'_{sr}$ is degraded by the set of missing concepts, non-satisfied and added concept constraints. In order to be compliant with the definition of the *sat* function described in the previous section and in order for $M_{sim}$ to produce correct results according to our assumptions about query similarity, we have to make an adaptation to the set-of-elements form of source queries when compared with target ones: the problem originates from additional joins on relation keys on rewritten queries; these new constraints do not correspond to any of previous rewritings or the original query itself; however, as stated in Section 3.1 we choose to consider these additional conditions as necessary and non-destructive to the original query. Since *sat* exports all concepts of the key join, it affects negatively $M_{sim}$. Therefore, when comparing such query versions we add to the source query *fake* condition elements that correspond to new key joins of the target query. Specifically, for each new key join $Cn.E_i = \{C_i, true, true\}$ we add to the source query the element $Cn.E_i = \{C_i, false, true\}$.

For example, the set-of-elements form of $Q_{StuartDB\_sr}$ and $Q_{LuDB\_sr}$ is:
$Q_{StuartDB\_sr} = \{\{\{Pid, true\}, \{DisDescr, true\}, \{Symptom, true\}, \{TreatDescr, true\}, \{Dosology, false\}\}, \{\{Visits.Did = Disease.Did, false, true\}, \{Disease.Did = Treatment.Did, false, true\}\}\}$
$Q_{LuDB\_sr} = \{\{\{Insurance\sharp, true\}, \{DisDescr, false\}, \{AvgFever, true\}, \{TreatDescr, false\}, \{Dosology, false\}\}, \{\{V.Did = D.Did, false, true\}, \{Disease.Did = Treatment.Did, false, true\}, \{Disease.Did = Patients.Did, true, true\}, \{Age < 13, true, true\}\}$.

Note, that the order of the elements in the $Q_{StuartDB\_sr}$, $Q_{LuDB\_sr}$ set is the same as in the $Q_{orig}$; thus, for example $Q_{LuDB\_sr}.Sl.\{Insurance\sharp, true\}$ corresponds $Q_{orig}.Sl.\{Pid, true\}$, and so on. It is obvious that one of the 'select' elements of $Q_{orig}$ are missing from $Q_{StuartDB\_sr}$ and two more from $Q_{LuDB\_sr}$. Also, $Q_{StuartDB\_sr}$ has satisfied the conditions of $Q_{orig}$ and no new ones, whereas $Q_{LuDB\_sr}$ has one new value condition, $\{Age < 13, true, true\}$ and one new key join, $\{Disease.Did = Paients.Did, true, true\}$. When $Q_{LuDB\_sr}$ is compared for similarity with $Q_{StuartDB\_sr}$, we add to the second a fake condition $\{Disease.Did = Patients.Did, false, true\}$.

**Definition 6.** *Considering two relational schemas $S$, $S'$ with a mapping between them $M(S,S') = \{Cr_M, Cond_M\}$, the successively rewritten version $Q'_{sr} = \{Sl', Cn'\}$ on $S'$ of a query $Q = \{Sl, Cn\}$ on $S$ based on $M(S,S')$ has:*
*- $Sl'.E'_i = Sl.E_i$ and $Cn'.E'_i = Cn.E_i$ if $pres_i = false$,*
*- for $Cn.E_i = \{C_i, true, true\}$, if $C_i \in Cond_M$, $Cn'.E_i = \{C_i, pres, true\}$, $pres = true$ or $pres = false$, else $Cn'.E_i = \{C_i, false, false\}$*
*- there are new condition elements $E_j = \{C_i, true, true\}$ added in $Q'_{sr}.Cn'$, where $C_i \in Cond_M \wedge \nexists E_i \in Cn$ such that $Cn.E_i = \{C_i, pres, st\}$*
*- for all the rest $Sl.E_i$, $Cn.E_i$ there are corresponding $Sl'.E_i$, $Cn'.E_i$ with $pres = true$, $st = true$ rewritten according to the rewriting algorithm.*

Note that a concept of schema $S$ can correspond to more than one concepts of schema $S'$. In this case, the query on $S$ is rewritten to a set of queries in $S'$, one for each such correspondence. The results of the queries are unified and returned to the query initiator. This is compliant with the query rewriting algorithms ( [25], [30] and [3]). For simplicity, in the rest of this work we assume that in a set of mappings $\mathcal{M}(S,S')$, there is only one correspondence for each schema concept $R.A \in S$ or $R.A \in S'$. The generalization of the clustering algorithm to union of queries is straightforward.

### 5.2 Description of the Clustering Process

In the following we describe the clustering process.

**Query Propagation:** Suppose that a query initiator $P_I$ initiates a query $Q_{orig} = \{Sl, Cn\}$. $P_I$ propagates to its acquaintees the set $\{Q_{orig}, w_{Q_{orig}}\}$, where $w_{Q_{orig}} = \{w_{Sl}, w_{Cn}\}$ are two sets of weights that refer to the 'select' attributes and the 'where' conditions. For example, the $w_{Sl} = \{w_{Sl_1}, .., w_{Sl_m}\}$ correspond to the members of $Sl = \{E_1, .., E_m\}$. The weights declare the importance of the respective query element, range in $(0,1]$ and add to the unit: $\sum w_{Sl_i} + \sum w_{Cn_j} = 1$. By default, the values of the weights conform to the criteria of section 3.2. However, the user may define weight values that are special to the posed query. Thus, high $w_i$ values indicate that $E_i.A_i/E_i.C_i$ plays an important role in the query and good rewritings of the latter are considered the accurate rewriting of $A_i/C_i$. On the other hand, low values of $w_i$ show that the query initiator is flexible concerning the rewriting of $A_i/C_i$, i.e., not so accurate rewritings of $A_i$ are accepted or that $C_i$ is not really important in the query semantics. For example, it is natural for join conditions to be considered important and have high weight values. Every peer on the query path forwards to the next acquaintee not only the successively rewritten version, but also the original query. Assume that $P$ is a peer on the query path and $Q_{sr\_P}$ is the version of $Q_{orig}$ that has been successively rewritten through peer mappings until it reached peer $P$ (including $P$). Assume also that the next peer on the query path is $P'$; then $P$ propagates to $P'$ the following set: $\{Q_{orig}, w_{Q_{orig}}, Q_{sr\_P}\}$. $P'$ has two options: it can either rewrite and answer $Q_{orig}$ or $Q_{sr\_P}$.

**Query Answering:** As aforementioned, all peers own a query reformulation mechanism that uses acquaintance mappings and query rewriting algorithms, but also a mechanism for automatic schema matching. Thus, $P'$ successively rewrites $Q_{sr\_P}$ to $Q_{sr\_P'}$ through the mappings between $P$ and $P'$. Next, $P'$ determines which elements of $Q_{orig}$ are not rewritten in $Q_{sr\_P'}$ and uses automatic schema matching to determine an ad hoc rewriting of them; the ad hoc rewritten elements are added to $Q_{sr\_P'}$ and produce an augmented version of the latter, $Q_{sra\_P'}$.

Moreover, $P'$ uses automatic matching to rewrite the $Q_{orig}$ from scratch to $Q_{ar\_P'}$. Hence, it can pick one of the two locally rewritten versions, $Q_{sra\_P'}$ and $Q_{ar\_P'}$ in order to answer $Q_{orig}$. The reason for considering an automatic rewritten version of the original query is to recover poorly successively rewritten elements: elements may be poorly successively rewritten at some node on the query path; if the mapping chain allows these elements to be successively rewritten further on, they may not have the chance to recover from the poor rewriting, even if there is such a possibility through automatic matching at some intermediate node. To demonstrate this, remember the example of Section 1, where a ‘select’ attribute of $Q_{orig}$ is ‘Ache’ and this is successively rewritten in *StuartDB* as ‘Symptom’ which is mapped to ‘AvgFever’ in the schema of *LuDB*. However, the latter comprises an attribute ‘Ache’ which fits best the respective element of $Q_{orig}$. Yet, if $Q_{orig}$ does not have the chance to be automatically matched and rewritten from scratch, the original ‘Ache’ will be rewritten to ‘AvgFever’ through the respective mapping.

$P'$ compares the two rewritten versions, $Q_{sra\_P'}$ and $Q_{ar\_P'}$, with $Q_{orig}$, in order to decide which one is more *similar* to original query, i.e., which version is semantically closer to the original query. For this, $P'$ uses a function $M_{sim_{P'}}(Q_{orig}, Q_{rewr})$ that measures the similarity between the source, $Q_{orig}$, and the rewritten query, $Q_{rewr}$. In other words, $M_{sim_{P'}}$ is the estimation of peer $P'$ about “how much” of $Q_{orig}$ can be answered by $Q_{rewr}$. $M_{sim_{P'}}$ is a function that quantifies the semantic similarity of the two queries.

Depending on the values of $M_{sim}(Q_{orig}, Q_{ar\_P'})$, $M_{sim}(Q_{orig}, Q_{sra\_P'})$, $P'$ answers the most information preserving version with respect to $Q_{orig}$. $P'$ replies to $P_I$ with a packet that carries the original query, $Q_{orig}$, and the answered version, i.e., the successively rewritten version augmented with automatic matching, $Q_{sra}$ or the automatic matched-rewritten version $Q_{ar}$, together with the resulted tuples, *Res*:

$$Ans_{P'}(Q_{orig}, Q_x, Res(Q_x)), \text{x = ar or sra} \tag{9}$$

**Answer Evaluation:** The query initiator $P_I$ evaluates the received answer with a function $Ev(Q_{orig}, Q_x, Res)$ that is based on $M_{sim_{P_I}}$ and can take into consideration the returned tuples $Res(Q_x)$. Note that $M_{sim_{P_I}}$ can be the same or a different function than $M_{sim_{P'}}$. *GrouPeer* aims at finding mutual interests of pairs of peers by allowing the latter to have their own perspective on query, schema and data semantics (that is expressed in a quantitative way through $M_{sim}$). Hence, each peer is allowed to judge the similarity of queries and the quality of received answers by its own means. We note that, at the moment *GrouPeer* does not provide a mechanism for evaluating the query answers, *Res*, automatically; this is a very difficult task and presumably it cannot be automated for general applications. Thus, *GrouPeer* by default evaluates the structural similarity of queries and respective answers. Yet, the evaluation function $Ev$ can be implemented in such a way so that the quality and appropriateness of *Res* can be derived in a proper way (for example, through human interaction).

$P_I$ replies to $P'$ with its estimation details about the reformulation of the answered query version: thus, $P_I$ sends to $P'$ the set:

$$Sat(Q_{orig}, Q_x) = \{sat(E_i, E_i') | \forall E_i \in Sl, Cn \wedge E_i' \in Sl', Cn'\}, \text{x = ar or sra} \tag{10}$$

which indicates the evaluation of $P_I$ for the satisfaction of each element in the original query by the respective element in the answered rewritten version. Added condition elements are not evaluated explicitly, since they are always considered as deviations from the original query. $P'$ can use these estimations in order to determine:

- for successively rewritten elements, if they were poorly rewritten and if they should be automatically rewritten.
- for automatically rewritten elements, how successful were the automatically produced mappings.

Using the above estimations, $P'$ will make its final choice in establishing a better mapping with $P_I$.

**Building Mappings:** $P'$ can use the $Sat(Q_{orig}, Q_x)$ values in order to make better decisions about query rewriting the next time it receives a query from $P_I$. $P'$ keeps bad and good estimations separately and gradually builds schema mappings with $P_I$, which gradually ameliorate automatic rewriting of query elements on $P'$ for queries initiated by $P_I$. Assuming that the schemas of peers $P_I$, $P'$ are $S_I$, $S'$, respectively, $P'$ keeps bad correspondences of concepts in the set $BCr(S_I, S')$. This set is augmented bad correspondences reported by $P_I$ to $P'$ after answer evaluations. $P'$ avoids correspondences in $BCr(S_I, S')$ when it attempts automatic matches on new incoming queries from $P_I$. Oppositely, $P'$ uses good answer evaluations of $P_I$ in order to produce mappings that will aid the automatic rewriting of new incoming queries of $P_I$. Actually, $P'$ uses pairs of incoming queries and their locally rewritten and answered versions, $\{Q_{orig}, Q_{rewr}\}$ as GLAV mappings, if the answer is evaluated positively by $P_I$. Furthermore, $P'$ merges mappings it holds for $P_I$ whenever this is possible. Specifically, two mappings $M1(S_I, S') = \{Cr_{M1}, Cond_{M1}\}$, $M2(S_I, S') = \{Cr_{M2}, Cond_{M2}\}$ are merged if the one is contained in the other, i.e. $Cr_{M1} \equiv Cr_{M2}$. In this case mappings are merged because the existence of both of them is pointless, since the one is more general than the other.Also, $M1$, $M2$ are merged if they have different concept correspondences that hold under the same constraints, i.e. $Cond_{M1} \equiv Cond_{M2}$, (see section 5.3). The merging of mappings in this case aims solely at their compression; thus, it can be omitted in favor of mapping accuracy. Of course the automatic merging of mappings could be more sophisticated and could produce GAV and LAV mappings. Yet, this is out of the scope of this work.

The produced mappings between $P_I$ and $P'$ can serve as peer mappings in case the two peers become acquainted.

**Acquaintance Establishment:** $P_I$ accumulates the $M_{sim_{P_I}}$ for answers coming from $P'$. When the average of these estimations overcomes a threshold $\theta_{P_I}$, then $P_I$ contacts $P'$ and asks the latter to become its acquaintee.

The threshold $\theta_{P_I}$ shows on behalf of $P_I$ the requirement for *information capacity* of peers in order for them to become direct neighbors. The term *information capacity* refers to the amount of information held in a database schema and is investigated in [20] and other works such as [29] in the context of data and schema integration.

In the context of data exchange in P2P databases, we use this term to refer to the capacity of the information in terms of semantics, i.e., concepts and their logical interrelations. Thus, we use the intersection of the information capacities (in terms of semantics) of two peers in order to decide if these should become acquainted. Since $M_{sim}$ measures the semantic relevance of two query schemas, we base the measurement of the intersection of the information capacities on accumulated $M_{sim}$ values. Therefore, the threshold $\theta_{P_I}$ refers to the lowest overall degree the common information capacities of two peers that is acceptable for the new acquaintance.

Next, we present in detail the algorithm of the described clustering procedure.

## 5.3 Clustering Algorithm

The algorithm includes two procedures running on each peer: one that runs for locally initiated queries and one that runs for incoming queries. Also, we briefly present the procedure that produces mappings between the query replier and the initiator, based on the original and reformulated version of the queries.

**Clustering Algorithm**
On each peer $P$ execute:
-the Query Initiator Procedure referring to $P$ as $P_I$
-the Query Replier Procedure referring to $P$ as $P'$

**Query Initiator Procedure**
*This procedure accumulates answers for locally initiated queries, sends evaluation feedback to responding nodes and invokes the acquaintance process for discovered similar peers*
Input: a set of candidate acquaintees $Acq = \bigcup Acq_i, Acq_i = \{P_i, Ev_{P_i}\}$; the $\theta_{P_I}$ threshold; a similarity measure $M_{sim_{P_I}}(Q, Q_r)$.

```
Initialization: $Acq = \emptyset$
**while** (online) **do**:
$\{$for each new locally initiated query $Q_{orig} = \{Sl, Cn\}$ and user-defined $w_{Q_{orig}}$ do:
**Step1**: propagate the set $\{Q_{orig}, w_{Q_{orig}}\}$
**Step2**: wait for answers to $Q_{orig}$
for each received packet $Ans_{P'}(Q_{orig}, Q_x, Res(Q_x))$ do:
- if $\{P', \_\} \notin Acq$ do $Acq = Acq \cup \{P', 0\}$
- update $P'$'s evaluation $Ev_{P'} = Ev_{P'} + M_{sim_{P_I}}(Q_{orig}, Q_x) \cdot fun(Res(Q_x))$
- send to $P'$ the set $sat(Q_{orig}, Q_x)$
- if $Ev_{P'} \geq \theta_{P_I}$ ask $P'$ to become an acquaintee $\}$
```

**Query Replier Procedure**

*This procedure answers incoming queries, receives evaluations for the answers and builds mappings with the respective remote peers*

```
Input: a set of candidate acquaintees $Acq = \bigcup Acq_i, Acq_i = \{P_i, \mathcal{M}(S_i, S'), BCr(S_i, S')\}$;
a similarity measure $M_{sim_{P'}}(Q, Q_r)$;
Initialization: $Acq = \emptyset$
**while** (online) **do**:
$\{$for each received query packet $\{Q_{orig}, w_{Q_{orig}}, Q_{sr\_P}\}$ originated from peer $P_I$ do:
**Step1**: if $P_I$ is new to $P'$ do $Acq \cup \{P_I, \emptyset, \emptyset\}$
**Step2**: automatically rewrite $Q_{orig}$ to $Q_{ar\_P'}$ using $\mathcal{M}(S_I, S')$ and avoiding $BCr(S_I, S')$; save the automatic mapping of $\{Q_{orig}, Q_{ar\_P'}\}$ as $M_{aut}(S_I, S')$
**Step3**: successively reformulate $Q_{sr\_P}$ to $Q_{sr\_P'}$ through the set of mappings $\mathcal{M}(S, S')$ between the schemas $S, S'$ of $P, P'$ respectively
**Step4**: - determine the set of elements that are not satisfied in $Q_{sr\_P'}$;
- produce the mapping $M_{sr}(S_I, S')$ using the ***Mapping Production Procedure*** corresponding to $\{Q_{orig}, Q_{sr\_P'}\}$;
- automatically rewrite the non-satisfied elements of $Q_{sr\_P'}$ using $M_{sr}(S_I, S') \cup \mathcal{M}(S_I, S')$ as input mappings and avoiding $BCr(S_I, S')$;
- produce the augmented $Q_{sra\_P'}$, save the automatically produced mapping as $M_{sra}(S_I, S')$:
- determine the set $\{Slf_{sr\_P'}, Cnf_{sr\_P'}\}$, where $Slf_{sr\_P'} = \bigcup Sl_{sr\_P'}.E_j$, $\forall Sl_{sr\_P'}.E_j \in Q_{sr\_P'}$ for which $pres = false$ and $Cnf_{sr\_P'} = \bigcup Cn_{sr\_P'}.E_j$, $\forall Cn_{sr\_P'}.E_j \in Q_{sr\_P'}$ for which $st = false$
- determine the set $\{Slt_{sr\_P'}, Cnt_{sr\_P'}\}$, where $Slt_{sr\_P'} = Sl_{sr\_P'} - Slf_{sr\_P'}$ and $Cnt_{sr\_P'} = Cn_{sr\_P'} - Cnf_{sr\_P'}$;
- produce the mapping $M_{sr}(S_I, S')$ corresponding to $\{Q_{orig}, Q_{sr\_P'}\}$;
- determine the set $\{Slf_{orig}, Cnf_{orig}\}$, where $Slf_{orig} = \bigcup Sl_{orig}.E_j$ for which the corresponding $Sl_{sr\_P}.E_j \in Slf_{sr\_P'}$ and $Cnf_{orig} = \bigcup Cn_{orig}.E_j$ for which the corresponding $Cn_{sr\_P}.E_j \in Cnf_{sr\_P'}$ ;
- automatically rewrite $Slf_{orig}$ to $Slf_{sra\_P'}$ and $Cnf_{orig}$ to $Cnf_{sra\_P'}$ using $M_{sr}(S_I, S') \cup \mathcal{M}(S_I, S')$ as input mappings; produce $Q_{sra\_P'} = \{Slt_{sr\_P'} \cup Slf_{sra\_P'}, Cnt_{sr\_P'} \cup Cnf_{sra\_P'}\}$ and save the automatic mapping of $\{Slf_{orig}, Slf_{sra\_P'}\}$, $\{Cnf_{orig}, Cnf_{sra\_P'}\}$ in $M_{sra}(S_I, S')$
**Step5**: calculate $M_{sim}(Q_{orig}, Q_{ar\_P'})$, $M_{sim}(Q_{orig}, Q_{sra\_P'})$; if $M_{sim}(Q_{orig}, Q_{ar\_P'}) > M_{sim}$ $(Q_{orig}, Q_{sra\_P'})$, produce $Res(Q_{ar\_P'})$ and set $\mathcal{M}(S_I, S') = \mathcal{M}(S_I, S') \cup M_{aut}(S_I, S')$ else produce $Res(Q_{sra\_P'})$ and set $\mathcal{M}(S_I, S') = \mathcal{M}(S_I, S') \cup M_{sr}(S_I, S') \cup M_{sra}(S_I, S')$
```

**Step6**: send to $P_I$ the packet $Ans_{P'}(Q_{orig}, Q_x, Res(Q_x))$, $x = ar\_P'$ or $x = sra\_P'$
**Step7**: receive from $P_I$ the set $Sat(Q_{orig}, Q_x)$; for the not satisfied elements, i.e. $sat(E_i, E_i') \neq \emptyset$, remove respective correspondences, $Cr$ from mappings in $M(S_I, S')$, i.e. $\forall M(S_I, S') = \{Cr_M, Cond_M\} \in \mathcal{M}(S_I, S')$ such that $Cr \in Cr_M$, do $Cr_M = Cr_M - Cr$; and add $Cr$ to $BCr(S_I, S')$
**Step8**: merge contained mappings in $\mathcal{M}(S_I, S')$ using the ***Mapping Merging Procedure***$\}$.

**Mapping Production Procedure**
Input: a pair of queries $\{Q_{orig}, Q_{sr\_P'}\}$, $Q_{orig} = \{Sl, Cn\}$, $Q_{sr\_P'} = \{Sl', Cn'\}$
Output: the mapping $M_{sr}(S_I, S') = \{Cr_M, Cond_M\}$.
Initialization: $M_{sr}(S_I, S') = \{\emptyset, \emptyset\}$
**Step1**: add to $Cr_{M_{sr}}$ new correspondences for pairs $\{Sl.E_i, Sl'.E_i'\}$: for each pair of elements $\{Sl.E_i, Sl'.E_i'\}$ with $E_i.pres = E_i'.pres = true$, create correspondence $Cr \equiv E_i.A = E_i'.A'$, and do $Cr_{M_{sr}} = Cr_{M_{sr}} \cup Cr$
**Step2**: update $Cr_{M_{sr}}$, $Cond_{M_{sr}}$ for pairs $\{Cn.E_i, Cn'.E_i'\}$: for each pair of elements $\{Cn.E_i, Cn'.E_i'\}$ with $E_i.pres = E_i'.pres = true$:
- if $E_i.C \equiv R.A = const$ and $E_i'.C \equiv R'.A' = const$ create correspondence $Cr \equiv R.A = R'.A'$, and do $Cond_{M_{sr}} = Cond_{M_{sr}} \cup E_i.C \cup E_i'.C$ and $Cr_{M_{sr}} = Cr_{M_{sr}} \cup R.A = R'.A'$
- if $E_i.C = (R_1.A_1 = R_2.A_2)$ and $E_i'.C = (R_1'.A_1' = R_2'.A_2')$ create the correspondences $Cr = \{R_1.A_1 = R_1'.A_1', R_2.A_2 = R_2'.A_2'\}$, and do $Cond_{M_{sr}} = Cond_{M_{sr}} \cup E_i.C \cup E_i'.C$ and $Cr_{M_{sr}} = Cr_{M_{sr}} \cup Cr$

**Mapping Merging Procedure**
Input: a set of mappings $\mathcal{M}(S_I, S') = \{Cr_M, Cond_M\}$
Output: the altered set $\mathcal{M}(S_I, S')$
$\{$for each pair $M1(S_I, S') = \{Cr_{M1}, Cond_{M1}\}, M2(S_I, S') = \{Cr_{M2}, Cond_{M2}\} \in \mathcal{M}(S_I, S')$ do:
**Step1**: merge mappings with $Cr_{M1} \equiv Cr_{M2}$: if $Cr_{M1} \equiv Cr_{M2}$ and $Cond_{M1} \cap Cond_{M2} \neq \emptyset$, merge $M1(S_I, S')$ and $M2(S_I, S')$ by setting $M1(S_I, S') = \{Cr_{M1}, Cond\}$, where $Cond = Cond_{M1}$ if $Cond_{M1} \subseteq Cond_{M2}$ or $Cond = Cond_{M2}$ elsewise; remove $M2(S_I, S')$ from $\mathcal{M}(S_I, S')$
**Step2**: merge mappings with $Cond_{M1} \equiv Cond_{M2}$: if $Cond_{M1} \equiv Cond_{M2}$ merge $M1(S_I, S')$ and $M2(S_I, S')$ by setting $M2(S_I, S') = \{Cr, Cond_{M2}\}$ where $Cr = Cr_{M1} \cup Cr_{M2}$; remove $M1(S_I, S')$ from $\mathcal{M}(S_I, S')$.$\}$.

At step 3 of the query replier procedure, the peer augments the successively reformulated query version using automatic rewriting. In order to do so, the concepts in $Q_{sr}'$ have to be translated into respective concepts in $Q_{orig}$, so that the mappings $\mathcal{M}(S_I, S')$ can be used as input to the automatic matcher and new mappings can be formed between $S_I$ and $S'$. Mappings $M_{sra}$ and $M_{aut}$ are produced by the peer's schema matching tool. Mappings $M_{sr}$ are created by the mapping production procedure that produces one mapping for each pair of input queries, $Q_{orig}$, $Q_{sr}$ : essentially the queries form the GLAV mapping $Q_{orig}$:-$Q_{sr}$ where only the present elements are involved.

**Example** Suppose that $Q_{orig}$ is translated to $Q_{LuDB\_ideal}$ with the obvious correspondences. Then, we obtain the following GLAV mapping:
$Mi_{DavisDB\_LuDB}$:
Visits$_{DavisDB}$(Pid, _, Did),Disease$_{DavisDB}$(Did, _, Ache), Treatment$_{DavisDB}$(Did, Drug, _):-Disease$_{LuDB}$(Did, _, Drug), Patients$_{LuDB}$(Insurance♯, Did, _, Ache), $\{Pid = Insurance\sharp\}$

When the query replier gets the evaluation of the initiator, it refreshes the existing mappings by removing bad correspondences. These ones are kept aside, in $BCr(S_I, S')$.

As a final step mappings are merged in order to avoid redundant matching information. Thus, mappings that are contained in others (with the classical meaning) are eliminated.

**Example** Remember the mapping:
$M_{StuartDB\_LuDB}$:
Treatment$_{StuartDB}$(Pid, _, _, Symptom, _, _):- Disease$_{LuDB}$(Did, AvgFever, _), Patients$_{LuDB}$ (Insurance♯, Did, _, _), Age $< 13$, $\{Pid = Insurance\sharp, Symptom = AvgFever\}$

Suppose that StuartDB and LuDB create a new mapping, that is similar to the above, but less restrictive:
$M'_{StuartDB\_LuDB}$:
Treatment$_{StuartDB}$(Pid, _, _, Symptom, _, _):- Disease$_{LuDB}$(Did, AvgFever, _), Patients$_{LuDB}$ (Insurance♯, Did, _, _), $\{Pid = Insurance\sharp, Symptom = AvgFever\}$

Then, the merging of $M_{StuartDB\_LuDB}$, $M'_{StuartDB\_LuDB}$ eliminates the first, since it is contained in the second.

Also, mappings with the same constraints are summarized in one mapping with the same constraints that contains all the respective correspondences. Again, suppose that StuartDB and LuDB have the mapping $M_{StuartDB\_LuDB}$ and they create the mapping:
$M''_{StuartDB\_LuDB}$:
Treatment$_{StuartDB}$(_, _, Symptom, _, _, _):- Disease$_{LuDB}$(Did, AvgFever, _), Patients$_{LuDB}$(_, Did, _, _), Age $< 13$, $\{Symptom = AvgFever\}$

The $M''_{StuartDB\_LuDB}$ is eliminated since $M_{StuartDB\_LuDB}$ contains the correspondence Symptom = AvgFever of the first.

## 5.4 Clustering Example

To give an example of the clustering process we refer to the situation of the motivating example in Section 1. Dr Davis poses the query $Q_{orig}$ which is successively rewritten as $Q_{StuartDB\_sr}$ on StuartDB and as $Q_{Lu\_sr}$ on LuDB. The successively rewritten version $Q_{LuDB\_sr}$ is augmented with the select attribute Disease.Drug using automatic matching. Thus, the augmented successively rewritten query on LuDB is the following:
$Q_{LuDB\_sra}$:

```
SELECT  P.Insurance#, D.AvgFever, D.Drug
FROM    Disease D, Patients P
WHERE   D.Did = P.Did, P.Age < 13
```

Also, the peer of Dr Lu uses automatic matching in order to rewrite from scratch the original query $Q_{orig}$. With automatic matching, the concept ‘Ache’ of $Q_{orig}$ is matched to ‘Ache’ on LuDB rather than ‘AvgFever’ which was the choice of successive rewriting. However, automatic matching fails to discover that the concepts ‘Pid’ and ‘Insurance♯’ both refer to the patients’ unique id. The produced query is:
$Q_{LuDB\_ar}$:

```
SELECT  D.Ache, D.Drug
FROM    Disease D, Patients P
WHERE   D.Did = P.Did
```

The Dr Lu peer calculates the value for the similarity function represented by equation (6) without weights for the rewritten versions:
$M_{sim}(Q_{orig}, Q_{LuDB\_sra})$ = 1-(2+1)/7 = 4/7,
$M_{sim}(Q_{orig}, Q_{LuDB\_ar})$ = 1- 3/7 = 4/7
The original query $Q_{orig}$ has 5 ’select’ attributes and 2 conditions. Thus, there are 7 query elements to be rewritten. $Q_{LuDB\_sra}$ 2 of the ’select’ attributes are not rewritten and there is 1 additional condition. In $Q_{LuDB\_ar}$ there are no additional conditions. Yet, 3 of the ’select’ attributes of $Q_{orig}$ are not rewritten. Note that the combination of the join constraints of $Q_{orig}$ is rewritten to a single join in $Q_{LuDB\_sra}$ and in $Q_{LuDB\_ar}$; thus, all of them are satisfied.

The peer decides to answer the $Q_{LuDB\_sra}$ version since it is considered safer than $Q_{LuDB\_ar}$ in terms of rewriting. Thus, it sends back to Dr Davis the answer packet:
$Ans_{LuDB}(Q_{orig}, Q_{LuDB\_sra}, Res(Q_{LuDB\_sra}))$,
The query initiator evaluates the received answer with the same $M_{sim}$ function (for simplicity we assume that the returned tuples are not taken into account). Thus, Dr Davis decides that the correspondence ‘Ache’ = ‘AvgFever’ is not satisfying; however, the rest of the correspondences seem fine. Dr Davis stores the similarity value $M_{sim}(Q_{orig}, Q_{LuDB\_sra})$ and replies to Dr Lu with the evaluation. Thus, the Dr Lu peer creates the first possible mapping with the peer of Dr Davis:
$M_{DavisDB\_LuDB}$:
Visits$_{DavisDB}$ (Pid, _, Did), Disease$_{DavisDB}$ (Did, _, _), Treatment$_{DavisDB}$ (Did, Drug, _):- Disease$_{LuDB}$(Did, _, Drug),Patients$_{LuDB}$ (Insurance♯, Did, _, _), $\{Pid = Insurance\sharp\}$

Note that the mapping created on LuDB refers to the entire schema of the latter but only to a portion of the schema encapsulated in $Q_{orig}$ of the remote peer of Dr Davis. Moreover, Dr Lu keeps ‘Ache’ = ‘AvgFever’ as a bad correspondence. Further on, let us assume that Dr Lu receives again the same $Q_{orig}$ initiated by Dr Davis through the peer of Dr Stuart. Again the successively rewritten version is the previous $Q_{LuDB\_sr}$. This time Dr Lu knows that ‘Ache’ is badly matched with ‘AvgFever’ and thus removes this correspondence from $Q_{LuDB\_sr}$. It concludes with the following augmented successively rewritten version:
$Q'_{LuDB\_sra}$:

```
SELECT  P.Insurance#, D.Ache, D.Drug
FROM    Disease D, Patients P
WHERE   D.Did = P.Did, P.Age < 13
```

Using the new mapping, the peer of Dr Lu rewrites $Q_{orig}$ automatically as:
$Q'_{LuDB\_ar}$:

```
SELECT  P.Insurance#, D.Ache, D.Drug
FROM    Disease D, Patients P
WHERE   D.Did = P.Did
```

Note that $Q'_{LuDB\_ar}$ is the query $Q_{LuDB\_ideal}$ described in Section 1 as the ideal translation of $Q_{orig}$ on LuDB. The peer of Dr Lu calculates the value for the similarity function (equation (6)) for the rewritten versions:
$M_{sim}(Q_{orig}, Q'_{LuDB\_sra}) = 4/7$, $M_{sim}(Q_{orig}, Q'_{LuDB\_ar}) = 5/7$
Dr Lu decides to answer the automatically rewritten version. Thus, it sends back to Dr Davis the answer packet:
$Ans_{LuDB}(Q_{orig}, Q'_{LuDB\_ar}, Res(Q'_{LuDB\_ar}))$. Dr Davis replies to Dr Lu that all matchings in the rewritten query are satisfying; Dr Davis decides that the average value of $M_{sim}$ of the answers by Dr Lu is enough for asking him to become his acquaintee. Dr Lu forms the mapping $M'_{DavisDB\_LuDB}$ of Section 1. The mapping merging procedure eliminates $M_{DavisDB\_LuDB}$ since $M'_{DavisDB\_LuDB}$ contains all the correspondences of the first and the two mappings have the same constraints. The two peers already have a mapping to start their acquaintance, $M'_{DavisDB\_LuDB}$.

## 5.5 Discussion on other *GrouPeer* Issues

In this section we discuss some details about the networking characteristics of *GrouPeer*. Furthermore, we discuss the usage of automatic schema matching in *GrouPeer* and the feasiblity of our approach.

**GrouPeer Protocol Internals** In the following we describe basic algorithm internals, specifically the query routing scheme and the addition/deletion of acquaintances.

*1) Routing:* Our method utilizes informed walks with a TTL parameter in order to propagate queries to nodes in the overlay. The requester deploys $k$ walkers, each following independent paths. A node forwards a query to the neighbor(s) whose schemas have the highest similarity value w.r.t. this query.

*2) Adding/dropping acquaintees:* We augment our clustering algorithm by allowing the dropping of existing neighbors in order to gradually improve on the random initial setup: New acquaintees are added whenever the local evaluation average is over $\theta_{P_I}$ and existing ones are dropped when its value is below $\theta_{P_I Low}$, provided we have received at least $THR$ replies from that node. This confidence parameter is important to ensure that the local evaluation is based on an adequate number of queries. We also define a maximum number of connections per peer, MAXDEGREE, which forces a neighbor addition to be preceded by the dropping of the neighbor with the smallest schema similarity if this limit is reached. A link is dropped whenever the local evaluation average is below $\theta_{P_I Low}$, provided the degrees of both nodes are at least MINDEGREE. This ensures that peers do not get disconnected from the network.

**Automatic Schema Matching** Schema matching is a fundamental issue in the database field, from database integration and warehousing to the newly proposed P2P data management systems. As discussed in [36], most approaches to this problem are semi-automatic, in that they assume human tuning of parameters and final refinement of the results. This is also the case in some recent P2P data management approaches (e.g., [33]).

However, in our context, schema matching assists query reformulation and its performance is boosted with increasing input mappings. Thus, it is feasible to use a tool that produces schema matches without human interference. Specifically, well-known matchers such as [10, 26, 34] and [9] benefit substantially from existing mappings and correspondences and are able to produce good matchings based on the first. In our case, successively rewritten query elements as well as mappings produced with earlier feedback from the query initiator can provide enough aid to the schema matching tool, so that it can discover further matchings. In addition, the fact that we limit our study to simple SPJ queries that do not require complicated rewritings, makes the problem easier to solve, (clearly, considering more complicated queries may even make the use of schema matching tools impossible). In general, our method does not expect automatic schema matching tools to discover more sophisticated relations other than simple correspondences.

Furthermore and in our context, automatic matchers are also used to rewrite queries from scratch, without any input mappings. In this case, we exploit the inherent capability of matchers to discover correspondences between closely semantically related words based on their internal general dictionaries. Accordingly, the role of matchers is to reveal a possible close semantic (and even structural) relation of the two schemas. For example, assume that two remote peers have almost the same schema. Then, an automatic matcher could certainly find out their schema similarity. Similarity between source and target schemas is observed in domain-specific applications. The reason is that: a) they store the same kind of data, b) there are specific policies for designing databases of specific domains and c) there are popular database products used in various fields. In our example, private doctors in general and specialty doctors in particular have to store the same kind of information, which is not of a wide variety: i.e., they care about listing their patients, their medical histories, their patients' visits, their own diagnosis and their own prescriptions for their patients. Moreover, it could be the case that some of these specialists use the same commercial tool to store their information. Obviously, for peer-databases with so similar schemas such as DavisDB and StuartDB, query rewriting can be done easily, even with speculations of the schema mapping.

## 6 Performance Evaluation

To evaluate the performance of *GrouPeer*, we use a message-level simulator written in C. By default, we randomly choose 100 nodes that play the role of the requesters, each making 100 queries to the system. We present results for 1,000-node random graphs (an adequate number of participants regarding our motivating application) with average node degrees around 4, created by the *BRITE* [27] topology generator. Results are averaged over 20 graphs of the same type and size, with 100 runs in each.

For the schemas stored at each node, we use two initial relational schemas, whose tables and attributes are uniformly distributed at nodes. The initial schema comprises 5 tables and 33 attributes. Seven attributes are keys with a total of 11 mappings (correspondences) between them. Each peer stores 10 random table columns (attributes) on average. Queries are generated randomly on the schema of each requester. They are formed on a single or multiple tables if applicable (join queries) by randomly selecting. Thus, peer similarity and schema/query similarity is not controlled. The kind of the initial schemas or the kind of queries that are used do not influence the experimental results. The latter can be influenced by parameters that affect the initial similarity of peer schemas and queries, i.e. the size of the initial schemas, the size of peer schemas and the size of the queries. We experimented with larger schemas (90 attributes over 12 tables) and a flat 100-attribute single table (no mappings between attributes). Because the creation of the individual schemas is computer-generated, an increase in the schema reduces the amount of the default similarity between nodes (unless more attributes are distributed per node). Nevertheless, the important observation is that, in all cases, *GrouPeer* maintains its relative advantages and behaves in a similar fashion.

Our basic performance metrics are the average similarity or *accuracy* of answers to the original queries (i.e., the structural similarity of the answered query over the original one evaluated at the requester), as well as the number of nodes that provide an answer. The accuracy is confined to the structural similarity since it aims at studying the effectiveness of the *GrouPeer* clustering process without human intervention.

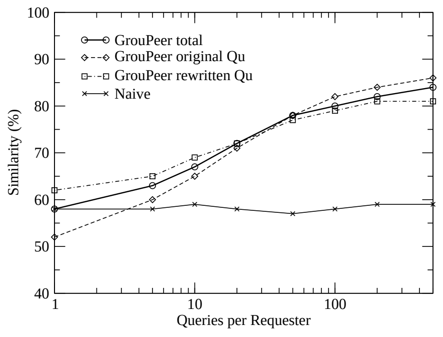
**Clustering Results** For the automatic rewriting of the original query, we simulate the possible erroneous outcome by altering the "perfect" rewriting by 50%. This is then gradually ameliorated through our learning process. We set the maximum number of allowed hops per query TTL=6, the number of deployed walkers $k = 3$, as well as $\theta_{P_I} = 0.7$ and $\theta_{P_I Low} = 0.3$ using a threshold parameter of *THR*=5 replies. Acquaintees can be dropped for nodes who have more than two neighbors. Finally, we assume that the returned tuples do not play any role to the answer evaluation.

In this study we present experiments that focus on parameters that can affect the relative clustering in a non-predictable way. These are: the number of queries per requester, the size of the queries, the number of requesters. Additionally, we check the clustering performance for queries that contain constraints (joins).

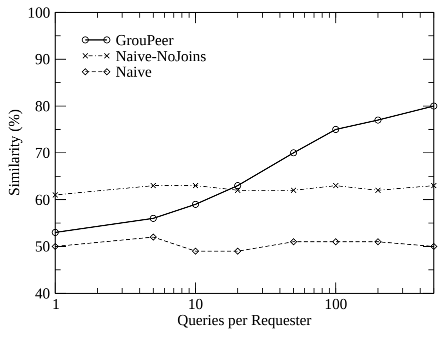
Figure 5(a) shows the performance of our algorithm by varying the number of queries posed by each of the 100 randomly selected requesters. Our method manages to return far more accurate results, achieving a similarity of around 85% in the steady state. The accuracy increases fast as more queries are created, since new acquaintees are added and neighbors with no contribution are dropped. We also present in more detail the experimental results for *GrouPeer* by analysing them in the respective values for answering the automatically rewritten ($Q_{ar}$ [16]) and the augmented rewritten versions of ($Q_{sra}$) the query. Both the automatically and the augmented consecutive

[16] Actually, since we do not assume the existence of an automatic matcher in the simulation other than the straightforward one that can produce id correspondences, the $Q_{ar}$ is reduced to the original query. Thus, the experimental results show that even for the worst situations of automatic matching, the clustering procedure performs very well.

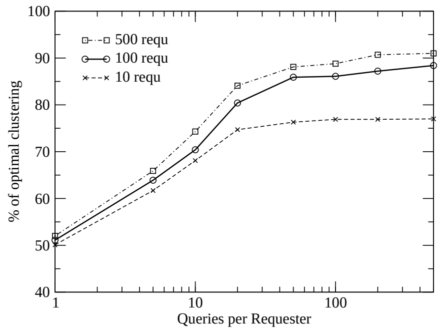
rewritten queries are answered with more precision. Our method's learning feature allows the automatic rewriting of the original query to improve over time as mappings are built between requester-replier pairs. Our clustering mechanism helps into bringing more information-rich nodes closer to requesters which also increases the accuracy of the consecutive rewritings. Our scheme is compared against *Naive*, which uses the same forwarding scheme as our method but answers only the successively rewritten query version. Our method can never fall below *Naive*'s performance but steadily performs better with more queries. Moreover, note that for 1 query per requester (thus, for an overlay on which no clustering is performed), answering the augmented consecutively rewritten version is still better than answering the plain consecutively rewritten version (*Naive*). Also, answering the original query, without classically rewriting it, gives low quality results without clustering the overlay. Finally, our scheme is almost as bandwidth-efficient as *Naive*, since the few additional messages reported are due to the communication between sources and requesters during the learning mechanism, as well as the message exchange when a new acquaintance is made.



(a) Similarity of answered versions to the original query for *GrouPeer* and *Naive* over variable queries per requester.



(b) Similarity of answered versions of join queries over variable queries per requester



(c) Ratio of *GrouPeer*'s clustering versus the optimal, given an equal number of acquaintees

**Fig. 5.** Experiments on the clustering procedure

**Table 1.** Performance varying the number of query attributes

| | **Similarity** | **Clustering** |
|---|---|---|
| $attr = 2, queries = 100$ | 0.87 | 80.2% |
| $attr = 2, queries = 500$ | 0.89 | 82.1% |
| $attr = 4, queries = 100$ | 0.80 | 86.1% |
| $attr = 4, queries = 500$ | 0.84 | 88.4% |
| $attr = 6, queries = 100$ | 0.71 | 83.0% |
| $attr = 6, queries = 500$ | 0.76 | 84.5% |
| $attr = 8, queries = 100$ | 0.67 | 80.0% |
| $attr = 8, queries = 500$ | 0.71 | 81.0% |

Next, we monitor *GrouPeer*'s performance by specifically tracking join queries in the same setting as the previous experiment. Figure 5(b) shows the results for our method and two different versions of *Naive*: The regular one we described before (which allows the rewriting of a join query even if the join is not mapped – like *GrouPeer*) and one that returns an empty query if the join(s) are not preserved. As before, we notice that *GrouPeer* performs at least as good as the original naive method and quickly increases in accurate answers as more queries are generated. The more strict naive method returns more similar results for few queries compared to our scheme. This happens as this method favors a complete (and thus more accurate) rewriting. Nevertheless, this comes at a cost of retrieving an answer from about 1/3 of the peers that *GrouPeer* gets answers from.

We also examine the quality of the clustering process as a means of locating nodes with similar schemas. For each requester, we measure the average similarity with its acquaintees at the end of the querying process and compare it with the best possible scenario: Having all top-*m* nodes in the overlay with schemas most similar to the initiator being its acquaintees, where *m* is equal to the total number of acquaintees this node has at the end of the querying process. We report the ratio of the actual average similarity to this optimal value in Figure 5(c).

Our methodology achieves clustering that is very close to the best achievable value in the steady state, while its quality quickly reaches that level. As more nodes become active, the process improves, since in *GrouPeer* nodes can take advantage of their neighbors' knowledge/connectivity. The ideal restructuring is hard to be achieved because of the random initial connectivity: The most similar nodes may not all receive queries and thus are not considered by the clustering process. Specifically, nodes may either be outside the query range or be left out of walkers' paths. By having more active nodes, our method effectively reduces the influence of the latter, since query initiators get replies by better nodes, taking advantage of other requesters' clustering. Figure 5(c) shows that in the steady state and with 10, 100 and 500 requesters, *GrouPeer* achieves 77%, 88% and 91% of the optimal clustering respectively. We can identify 88% of the optimal nodes in the entire network by having only 10% active nodes and each of them contacting at most $k \times TTL = 18$ nodes per query (this amounts to less than 2% of the peers).

Table 1 summarizes the performance of *GrouPeer* with a different number of query attributes (each requester making 100 or 500 queries). As the number or attributes per query increases, the accuracy of the answers slightly drops, since a smaller percentage of attributes has the chance to be satisfied. Note that the quality of the clustering increases up to a point, after which it starts to slightly decrease. This is due to the fact that there are two competing factors that affect the clustering process: The more attributes in a query, the more precise the clustering process becomes, since the initiator learns more information for its schema as a whole; the query similarity (which affects clustering through the $Ev$ function), on the other hand, decreases with the number of attributes.

We tested our method in graphs of different sizes (from 100 to 4K nodes) and different connectivities (power-law). Results of these runs are qualitatively similar to the presented ones.

## 7 Related Work

The Chatty Web [1] considers P2P systems that share semi-structured or structured information. The authors are concerned about the gradual degradation, in terms of syntax and semantics, of a query that is propagated along a network path. However, the Chatty Web approach considers peers that own very simple relational schemas and GAV mappings with their acquaintees. Instead, we are interested in more complex peer schemas and we consider GAV, LAV or GLAV mappings.

In [42], the authors propose optimization techniques for query reformulation in P2P data management systems. They focus on minimizing the rewriting of a query and pruning the respective propagation path in order to avoid redundant reformulations. Additionally, it is indicated that pre-computation of the query reformulation path-tree proves to accelerate the reformulation procedure despite the disadvantage of the necessary maintenance of pre-computed mappings. Our approach is specifically designed for large-scale unstructured overlays. First, it evades reformulation at peers poor in query-relevant information by adaptively choosing the version of the query to be answered. Moreover, while the work in [42] requires central knowledge of the system structure, our scheme enables nodes to operate in a completely decentralized fashion, utilizing the standard lookup operations to refine their local knowledge.

PeerDB [33] facilitates relational data sharing without any schema knowledge. Query matching and rewriting is based on keywords (provided by the users). A two-step process is described: First all nodes within a TTL radius are contacted, returning prospective answer meta-data. Then the user selects the ones that are relevant to the local query and the requester directly contacts the selected sources and asks for the results to the various rewritten versions of the query. Instead, our approach employs an automated technique based on a combination of successive query rewriting and query-schema matching, while it utilizes bandwidth-efficient walks instead of the costly flooding scheme.

The works in [18] and [21] deal with data exchange between peers. Ref. [18] presents a significant approach to the heterogeneity issue in P2P data management and proposes a language for schema mediation between peers. Also, the authors present an algorithm for query reformulation based on local-as-view as well as global-as-view query answering. In [21], the authors describe mechanisms for the declaration of data exchange policies on-the-fly based on ECA rules. They also propose a general architecture for peer-databases and elaborate on the establishment and abolishment of acquaintances between peers.

Beyond the above significant works, there are plenty that have talked about semantics and semantic clustering of peers. The work in [8] is one of the first to consider semantics in P2P systems and suggest the construction of semantic overlay networks, i.e. SONs. Later on, other researchers have attempted to go beyond the a priori static formulation of SONs: the work in [39] suggests the dynamic construction of the interest-based shortcuts in order for peers to route queries to nodes that are likely more capable of answering them. Inspired by [39], the authors in [44] but also in [19] exploit implicit approaches for discovering semantic proximity based on the history of query answering and the least recently used nodes. In the same spirit the work in [12] presents preliminary results about the clustering of the workload on the real popular systems e-Donkey and Kazaa.

Some of the well-known projects that have dealt with the data heterogeneity problem in P2P systems are [2, 16, 32, 41]. Edutella [32] is a schema-based network that holds RDF data. Peers have services (e.g. quering, mapping, mediating etc) that they share with other peers. Peers can formulate complex queries that are translated in wrappers to queries on the Edutella Common Data Model. Peers register their services and the kinds of queries they can answer to mediators. The lattter route the incoming queries to peers that are probably able to answer them. Edutella is an interesting effort towards the solution of the heterogeneity problem both of data and services. However, it is not focused on semantic clustering of peers and does not propose sophisticated methods for distributing queries to semantically relevant peers.

GridVine [2] is another project worth of attention. Based on a structured (i.e. implementing a DHT algorithm) P2P overlay network, P-Grid, GridVine achieves the management and mapping of complex data and schemas of meta-data. Specifically, RDF data and schemas are hashed and indexed in peers. P-Grid peers refer to a common underlying tree structure of characters. Each peer is associated with a tree leaf, and thus, with a string. GridVine allows schema inheritance and the creation and index of translation links that map pairs of schemas. Peers query RDF triples. Using the mapping links, queries are iteretively or recursively forwarded to peers that can answer them. Although GridVine is an interesting approach and offers many features related to semantics, the efficiency of the search algorithm is based on the underlying DHT, thus the structured form of the overlay, and not to semantic clustering of peers.

Similarly, pSearch [41] is a project that employs a DHT algorithm to build a solution to the problem of data semantic diversity in peers. Specifically, pSearch creates a semantic overlay by mapping overlay nodes to physical nodes in a CAN [38]. Documents as well as queries of peers are represented as semantic vectors. These are the keys to store, index or search the documents in CAN. The Vector Space Model (VSM) and the Latent Semantic Indexing (LSI) are used to create the semantic vectors. As GridVine, pSearch bases search efficiency on the structured form of the overlay, and, thus, does not solve the semantic diversity problem in an unstructured P2P system. Another disadvantage is that documents of newly-joined peers, with terms that are not encapsulated in the existing vector, cannot be indexed by them.

Finally, Bibster [16] is a project that exploits ontologies in order to enable P2P sharing of bibliographic data. Ontologies are used for importing data, formulating and routing queries and processing answers. Peers advertise their expertise and learn through ontologies about peers with similar data and interests.

Beyond semantic clustering, the work in [37] looks into the problem of discovering connectivity clusters of nodes in P2P networks, detecting the transmission of the same query multiple times at the same node.

## 8 Summary

In this paper we described *GrouPeer*, a methodology to solve the query degradation problem in P2P data management systems in the absence of global schema information. The key characteristic of our method is to allow peers to select the appropriate rewritten version of the query to answer. Incorporating efficient feedback between query initiators and content providers, we achieve the discovery of remote peers on query propagation paths that are rich in interesting information but veiled by poor path predecessors.

In effect, with the described procedure we manage to surpass the boundaries of successive query rewriting and reach hidden peers pertinent in requested information. Moreover, we have achieved the gradual training of remote peers in order to ameliorate query rewriting and give more accurate answers. Beyond this, peers develop candidate mappings with remote peers using their feedback about the quality of query answers. These mappings facilitate the possible acquaintance procedure between the respective peers. Nevertheless, all these benefits have been achieved through the sole exploitation of queries posed in the system. Without any additional processes or metadata, peers are enabled to discover remote peers with interesting information.

During this work, we also discussed and proposed techniques to tackle the important issues of query rewriting and query similarity in the context of unstructured P2P database systems.

In*GrouPeer* ‘active’ peers, in terms of the number of initiated queries and the number of answered ones, are compensated more with information about remote peers, than inactive ones. Consequently, the P2P overlay is progressively clustered in groups of peers with similar interests.

Experimental results show that *GrouPeer* nodes quickly identify the vast majority of best available peers by contacting only a very small number of peers per query. The clustering process effectively increases the quality of the returned results. Actually, clustering according to interests of peers benefits the successively query rewriting procedure, since peers on query paths are steadily ordered according to schema similarity. In addition, successively rewritten queries have the chance to travel longer paths before being totally degraded due to poor peer mappings. The outcome is better quality of peer query answers.

Currently, we extend *GrouPeer* with a ’grouping’ technique that can follow the clustering process. Actually, the grouping process intends to create explicit groups from implicit clusters of peers. Specifically, the grouping process will create a schema that is representative of the interests of the peers that belong to each cluster. The groups (and therefore the group schemas) can be used in order to ameliorate even more the

the quality of query answering but also facilitate the join of new peers in the groups according to their interests.

In the near future we intend to implement *GrouPeer* as a real system and experiment with the clustering technique as well as with the query similarity and reformulation approaches on real data and situations.

Finally, in the future we intend to apply *GrouPeer* to P2P overlays that handle data other than relational. We will focus on data that conform to models with more expressive power, such as the 'Resource Description Framework' (i.e. *RDF*). We presume that the potential of the *GrouPeer* approach will become more apparent for data with rich semantics.

## ACKNOWLEDGMENTS

This research has been partly supported by the European Union - European Social Fund & National Resources under the PENED/EPAN program (Greek Secretariat of Research and Technology)

## References

[1] K. Aberer, P. Cudre-Mauroux, and M. Hauswirth. The Chatty Web: Emergent Semantics Through Gossiping. In *WWW Conference*, 2003.

[2] K. Aberer, P. Cudre-Mauroux, M. Hauswirth, and T. Van Pelt. Gridvine:Building internet-scale semantic overlay networks. In *International Semantic Web Conference*, 2004.

[3] F. Afrati, C. Li, and P. Mitra. Answering Queries Using Views with Arithmentic Comparisons. In *21th ACM PODS*, 2002.

[4] S. Agrawal, S. Chaudhuri, and G. Das. DBXplorer: A System for Keyword-Based Search over Relational Databases. In *ICDE*, 2002.

[5] S. Agrawal, S. Chaudhuri, G. Das, and A. Gionis. Automated Ranking of Database Query Results. In *CIDR*, 2003.

[6] W. W. Chu and G. Zhang. Associative Query Answering via Query Feature Similarity. In *IIS*, 1997.

[7] W. Cohen. Integration of Heterogeneous Databases Without Common Domains Using Queries Based on Textual Similarity. In *SIGMOD*, 1998.

[8] A. Crespo and H. Garcia-Molina. Semantic Overlay Networks for P2P Systems. In *Technical Report*, 2003.

[9] R. Dhamankar, Y. Lee, A. Doan, A. Halevy, and P. Domingos. iMAP: Discovering Complex Semantic Matches between Database Schemas. In *ACM SIGMOD*, 2004.

[10] H. Do and E. Rahm. Coma - A System for Flexible Combination of Schema Matching Approaches. In *VLDB*, 2002.

[11] A. Doan, J. Madhavan, R. Dhamankar, P. Domingos, and A. Halevy. Learning to Match Ontologies on the Semantic Web. In *VLDB Journal*, 2003.

[12] F. Le Fessant, S. Handurukande, A.-M. Kermarrec, and L. Massoulie. Clustering in Peer-to-Peer File Sharing WorkLoads. In *IPTPS*, 2004.

[13] M. Friedman, A. Y. Levy, and T. D. Millstein. Navigational Plans for Data Integration. In *VLDB*, 2002.

[14] N. Fuhr. A Probabilistic Framework for Vague Queries and Imprecise Information in Databases. In *VLDB*, 1990.

[15] A. Ghosh, J. Parikh, V. S. Sengar, and J. R. Haritsa. Plan Selection based on Query Clustering. In *Intelligent Information Integration*, 1999.

[16] P. Haase, B. Schnizler, J. Broekstra, M. Ehrig, F. van Harmelen, M. Menken, P. Mika, M. Plechawski, P. Pyszlak, R. Siebes, S. Staab, and C. Tempich. Bibster - a semantics-based bibliographic peer-to-peer system. In *Journal of Web Semantics*, 2005.

[17] A. Halevy, Z. Ives, J. Madhavan, P. Mork, D. Suciu, and I. Tatarinov. The Piazza Peer Data Management System. In *IEEE Transactions on Knowledge and Data Engineering*, 2003.

[18] A. Halevy, Z. Ives, D. Suciu, and I. Tatarinov. Schema Mediation in Peer Data Management Systems. In *ICDE*, 2003.

[19] S. Handurukande, A.-M. Kermarrec, F. Le Fessant, and L. Massoulie. Exploiting Semantic Clustering in the eDonkey P2P Network. In *ACM SIGOPS*, 2004.

[20] R. Hull. Relative Information Capacity of Simple Relational Database Schemata. In *SIAM Journal of Computing*, 1986.

[21] V. Kantere, I. Kiringa, J. Mylopoulos, A. Kementsientidis, and M. Arenas. Coordinating P2P Databases Using ECA Rules. In *DBISP2P*, 2003.

[22] W. Kießling and G. Kostner. Preference SQL - Design, Implementation, Experiences . In *VLDB*, 2002.

[23] M. Lenzerini. Data Integration: A Theoretical Perspective. In *21th ACM PODS*, 2002.

[24] A. Y. Levy. Answering Queries Using Views: A Survey. In *VLDB Journal*, 2001.

[25] A. Y. Levy, A. Rajaraman, and J. O. Ordille. Query-answering Algorithms for Information Agents. In *13th International Conference on Artificila Intelligence*, 1996.

[26] J. Madhavan, P.Bernstein, and E. Rahm. Generic Schema Matching with Cupid. In *VLDB*, 2001.

[27] A. Medina, A. Lakhina, I. Matta, and J. Byers. BRITE: An Approach to Universal Topology Generation. In *MASCOTS*, 2001.

[28] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity Flooding: A versatile Graph Matching Algorithm and its Application to Schema Mathcing. In *ICDE*, 2002.

[29] J. Miller, Y. E. Ioannidis, and R. Ramakrishnan. The Use of Information Capacity in Schema Integration and Translation. In *VLDB*, 1993.

[30] P. Mitra. An Algorithm for Answering Queries Efficiently Using Views. In *Australesian Database Conference*, 2001.

[31] A. Motro. VAGUE: A User Interface to Relational Databases that Permis Vague Queries. In *TOIS 6(3), 187-214*, 1988.

[32] W. Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmer, and T. Risch. Edutella: A p2p networking infrastructure based on rdf. In *WWW*, 2002.

[33] W. S. Ng, B. C. Ooi, K.-L. Tan, and A. Zhou. PeerDB: A P2P-based System for Distributed Data Sharing. In *ICDE*, 2003.

[34] L. Popa, Y. Velegrakis, M. Hernandez, R. J. Miller, and R. Fagin. Translating Web Data. In *VLDB*, 2002.

[35] R. Pottinger and A. Levy. A Scalable Algorithm for Answering Queries Using Views. In *VLDB*, 2000.

[36] E. Rahm and P.Bernstein. A Survey of Approaches to Automatic Schema Matching. In *VLDB Journal*, 2001.

[37] Lakshmish Ramaswamy, Bugra Gedik, and Ling Liu. A Distributed Approach to Node Clustering in Decentralized Peer-to-Peer Networks. In *IEEE Transactions on Parallel and Distributed Systems*, 2005.

[38] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content Addressable Network. In *SIGCOMM*, 2001.

[39] K. Sripanidkulchai, B. Maggs, and H. Zhang. Efficient Content Location Using Interest-Based Locality in Peer-to-Peer Systems. In *INFOCOM*, 2003.

[40] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *SIGCOMM*, 2001.

[41] C. Tang, Z. Xu, and S. Dwarkadas. Peer-to-peer information retrieval using self-organizing semantic overlay networks. In *SIGCOMM*, 2003.

[42] I. Tatarinov and A.Halevy. Efficient Query Reformulation in Peer-Data Management Systems. In *SIGMOD*, 2004.

[43] J. D. Ullman. Principles of Database and Knowledge - Base Systems. In *Computer Science Press*, 1988.

[44] S. Voulgaris, A.-M. Kermarrec, L. Massoulie, and M. van Steen. Exploiting Semanntic Proximity in Peer-to-Peer Content Searching. In *FTDCS*, 2004.