Optimizing monitoring queries over distributed data

Optimizing Monitoring Queries over Distributed Data

Frank Neven[1] and Dieter Van de Craen*,[1]

[1] Hasselt University
{frank.neven,dieter.vandecraen}@uhasselt.be
Agoralaan, 3590 Diepenbeek, Belgium

Abstract. Scientific data in the life sciences is distributed over various independent multi-format databases and is constantly expanding. We discuss a scenario where a life science research lab monitors over time the results of queries to remote databases beyond their control. Queries are registered at a local system and get executed on a daily basis in batch mode. The goal of the paper is to study evaluation strategies minimizing the total number of accesses to databases when evaluating all queries in bulk. We use an abstraction based on the relational model with fan-out constraints and conjunctive queries. We show that the above problem remains NP-hard in two restricted settings: queries of bounded depth and the scenario with a fixed schema. We further show that both restrictions taken together results in a tractable problem. As the constant for the latter algorithm is too high to be feasible in practice, we present four heuristic methods that are experimentally compared on randomly generated and biologically motivated schemas. Our algorithms are based on a greedy method and approximations for the shortest common super sequence problem.

1 Introduction

In the field of the life sciences, scientific data is distributed over various web sites and data is mostly accessible via browsers or in some cases through primitive web services [13]. A characteristic of biological data is that it is abundantly available and rapidly growing. For instance, the daily updates to Genbank [5] alone range in size from 40 till 200 Megabytes. Therefore, the answers to searches may vary over time as more data becomes available. However, due to the limited access to the distributed sources it is cumbersome to repeat searches over time especially if they combine information from several web sites. In this paper, we consider the setting of a light-weight monitoring system that runs at a research lab where users can register queries which are executed periodically. Users are then notified when new answers to their queries arrive.

We give an example of the kind of queries biologists would like to monitor over time: a certain biological experiment on a rare organism results in a set

* Contact author

of genes. A search of the available data reveals a set of related genes in other organisms. However, only for a very small fraction of those genes their function is known. The researcher therefore would like to be notified when more information on the function of these genes becomes available. As shown in Example 2, such a query combines information from three sites: Genbank [5] containing gene related info and references to corresponding proteins, SwissProt [3] containing protein related info and some links to GO-entries, and GO [2] containing functional descriptions of proteins.

Although most popular biological websites and web services can be freely accessed, there are restrictions on the number of accesses and the amount of data that can be transferred per request. Furthermore, most data is transmitted using the HTTP-protocol, which makes the connection setup cost much higher than the data transfer cost. One single connection that transfers a lot of data is preferable to several smaller connections each transferring small amounts of data [18]. It is therefore of prime concern to combine queries and minimize the number of different communications. The goal of the paper is to study the latter problem.

We model a situation where a limited number of sites is available: rather twenty than hundreds or thousands as for instance is the case for peer-to-peer computing. Further, we consider a light-weight system and assume at most a few thousands of registered queries. Queries also have to be re-executed from scratch as there usually is no access to the updates the web sites receive. Although in practice most data is stored as flat files, we assume a relational view on this data and use conjunctive queries as a query language. This means that the actual queries should then be translated to appropriate calls to the web services or into HTTP-requests. We choose for this kind of abstraction rather than, for instance, going through an XML query language, as the focus of the paper is on minimizing communication not on the actual form of queries. Furthermore, the formalism of the relational model and conjunctive queries is sufficiently general to capture large parts of the available data and path-like search queries as described in the above scenario. On the other hand, the approach is specific enough to be translated into any reasonable query language or model.

We only allow an evaluation protocol for a set of queries to send a constant number of messages, where every message is a query or the transfer of a bounded number of tuples. We refer to these as bounded protocols. In Section 3, we require that messages are of size logarithmic in the size of the data which amounts to the same requirement. To allow queries to satisfy this requirement, schemas impose fan-out constraints which are for instance determined by domain experts (cf. Section 2).

We summarize the main results of the paper:

1. Not every conjunctive query can be evaluated by a bounded protocol. We show in Section 3 that deciding whether a set of queries can be evaluated by a bounded protocol is in polynomial time.
2. Minimizing the number of communications to simultaneously evaluate a set of queries is NP-complete. In Section 4, we show hardness for two restricted

cases: (1) queries of bounded depth (cf. Section 3); and, (2), queries over a fixed database schema. We use reductions from the Feedback Vertex Set (FVS) [8] and the Shortest Common supersequence problem (SCS) [15] which are known to be NP-hard. Furthermore, we show that both restrictions taken together results in a tractable problem which, unfortunately, is not practically useful due to the large constant.

3. We present four heuristic methods in Section 5. The first method is greedy-based. The other methods are based on approximations for SCS. Our experiments show that over random database schemas the Pairwise SCS performs best, while over a concrete biologically motivated database schema the greedy-method outperforms the rest in finding the best evaluation strategy. Finally, we remark that our experiments show that using our heuristics for 1000 random queries (consisting of 10 atoms each) on average only around 50 accesses to websites are necessary.

Due to space constraints, some proofs are omitted.

Related Work. The above described monitoring system is different from the usual publish/subscribe systems where users can specify by means of patterns what type of messages they are interested in. Such systems usually focus on new data only and can account for millions of subscribers [1]. A well known publish/subscribe system for biological researchers is PubCrawler [9] which scans daily updates to PubMed and Genbank, it keeps researchers informed on the current contents of PubMed and Genbank. Our setting allows for more advanced querying rather than keyword searching. A distributed database system consists of a single distributed DBMS. This DBMS manages multiple databases. A number of classes of distributed query optimization problems are known to be NP-complete [22]. They do not consider the setting of bounded communication. Heuristic methods were therefore developed to deal with these problems. An example of such an heuristic method is the query optimizer of SDD-1, which uses a greedy approach to find the semijoin order [4]. A multi database system supports operations on multiple heterogeneous local databases [17]. The most distinctive features of multi database systems are site autonomy and heterogeneity [14]. Distributed query optimization and multi database query optimization are distinctively different problems [14]. Our problem shows the strongest resemblance to the multi database query optimization. However, in our situation we have to evaluate multiple queries while using a minimal number of communications and a bounded number of tuples. In multi-query optimization the aim is to optimize in parallel a set of queries [16]. In contrast with our setting they do not consider minimizing the number of communications, also the number of considered queries is small (e.g., 5).

Acknowledgments. We thank Ivy Jansen for her suggestion to use and the creation of the box plots in Figures 1 and 2, Kerstin Koch for her help in constructing the biological schema, and, Stijn Vansummeren and Dan Suciu for their helpful comments on a previous draft of this paper.

2 Definitions

In this section, we present the necessary background and definitions. To keep our exposition simple, we model every biological website or source by one relation (as opposed to several relations which would be more realistic). It is straightforward to adapt our results to that setting. The distributed sources are hence modeled by a set of relation names in the understanding that every relation resides at a different site.

We assume an infinite set of relation names $\mathcal{R}$ and attribute names $\mathcal{A}$ with $\mathcal{R} \cap \mathcal{A} = \emptyset$. Every relation name has an associated finite set of attributes, denoted by $\mathrm{Att}(R)$. Let $\mathbf{D} = \{\mathbf{d_1}, \mathbf{d_2}, \ldots\}$ be an infinite domain of data values. An R-tuple t is a function from $\mathrm{Att}(R)$ to $\mathbf{D}$. An R-relation $R^{\mathbf{D}}$ is a finite set of R-tuples. The cardinality of a relation R, denoted by $|R|$, is the number of tuples in R. The size of a relation, denoted by $||R||$, is $k \times |R|$ where k is the number of attributes of R.

A *distributed schema* $(\mathcal{S}, \Delta)$ is a set of relation names $\mathcal{S}$ with associated attributes together with a set of fan-out constraints Δ defined below. A *database* $\mathcal{D}$ over $\mathcal{S}$ assigns an R-relation to every relation name R in $\mathcal{S}$. In the sequel we do not distinguish between the relation name R and the R-relation itself: we denote both of them by R. Denote by $\mathrm{DB}(\mathcal{S})$ the class of all databases over $\mathcal{S}$. To emphasize that the various relations in $\mathcal{S}$ are distributed we refer to them as *sites* or *sources*.

A *fan-out constraint* is a rule of the form $R : X \rightarrow_k Y$, where $X, Y \subseteq \mathrm{Att}(R)$. A database $\mathcal{D}$ satisfies a set of fan-out constraints Δ, denoted $\mathcal{D} \models \Delta$, iff for every rule $R : X \rightarrow_k Y \in \Delta$ and every tuple t, $|\pi_Y(\sigma_{X=t(X)}(R))| \leq k$. Here, π and σ are the relational operators denoting projection and selection. By $X = t(X)$, we abuse notation and mean $\bigwedge_{i=1}^{\ell} A_i = t(A_i)$ for $X = \{A_1, \ldots, A_\ell\}$. Intuitively, the constraint says that in R for every fixed value for the attributes in X there are at most k different values for the attributes in Y. In the sequel, we do not care about the actual value of k and simply write $R : X \rightarrow Y$ rather than $R : X \rightarrow_k Y$ to denote that there is some bound.

Example 1. Consider the following relational schema constituting four sites:

$$\begin{array}{l} \mathrm{Genbank}(\mathrm{gene\_id}, \mathrm{protein\_id}, \mathrm{organism}), \\ \mathrm{SwissProt}(\mathrm{protein\_id}, \mathrm{go\_id}, \mathrm{organism}), \\ \mathrm{Go}(\mathrm{go\_id}, \mathrm{name}), \text{ and} \\ \mathrm{Kegg}(\mathrm{pathway\_id}, \mathrm{protein\_id}). \end{array}$$

A tuple in Genbank contains a gene_id representing the id of the gene at hand. Every gene corresponds to one or more proteins which are listed in the SwissProt database by protein_id. The third component is the organism from which the gene originates, e.g., human, mouse, rat, ... Go is a database/ontology that contains function descriptions of proteins, e.g., serine protease. Only for a very limited number of proteins a functional description is actually known. Kegg contains information on pathways where special proteins are involved. We have

the following fan-out constraints:

$$\begin{array}{l}\text{Genbank} : \text{gene\_id} \rightarrow \text{protein\_id}, \text{organism}\\ \text{Genbank} : \text{protein\_id} \rightarrow \text{gene\_id}\\ \text{SwissProt} : \text{protein\_id} \rightarrow \text{go\_id}, \text{organism}\\ \text{Go} : \text{go\_id} \rightarrow \text{name}\\ \text{Kegg} : \text{pathway\_id} \rightarrow \text{protein\_id}\end{array}$$

Note that these are not necessarily keys. For instance, a gene_id can correspond to several protein_ids. The above relations are crude abstractions of existing sites [5, 3, 2, 11]. In [20] a more elaborate abstraction is given which we used for our experiments. □

As a query language, we employ the well-known formalism of *conjunctive queries*. An atom L is an expression $R(A_1 : x_1, \ldots, A_n : x_n)$, where R is a relation symbol, $A_i \in \text{Att}(R)$ and x_i is a variable or a data value for $i = 1, \ldots, n$. We require that $A_i \neq A_j$ for all $i \neq j$. Note that $\{A_1, \ldots, A_n\}$ need not be equal to $\text{Att}(R)$. A *variable assignment* ρ for L is a mapping that assigns to each variable in L a data value in $\mathbf{D}$. The atom $L = R(A_1 : x_1, \ldots, A_n : x_n)$ holds in $\mathcal{D}$ under ρ, denoted $\mathcal{D} \models L[\rho]$, iff there is a tuple $t \in R$ such that for every i, $t(A_i) = \rho(x_i)$.

A *conjunctive query* is then an expressions of the form

$$Q(X_1 : x_1, \ldots, X_k : x_k) \leftarrow L_1, \ldots, L_n,$$

where each L_i is an atom and each x_i occurs in at least one atom. The semantics is the usual one: Q defines the relation $Q(\mathcal{D}) = \{(X_1 : \rho(x_1), \ldots, X_n : \rho(x_n)) \mid \rho \text{ is an assignment s.t. } \forall i, \mathcal{D} \models L_i(\rho)\}$. The relational schema associated to $Q(\mathcal{D})$ consists of the single relation symbol Q where $\text{Att}(Q) := \{X_1, \ldots, X_n\}$.

The *size of an atom* is equal to the number of variables appearing in the atom. The *size of a query* is the sum of the sizes of its atoms.

Example 2. Consider the query

$$\begin{array}{rl}Q \leftarrow & \text{Genbank}(\text{gene\_id} : \text{'AC04654'}, \text{protein\_id} : x),\\ & \text{SwissProt}(\text{protein\_id} : x, \text{go\_id} : y),\\ & \text{Go}(\text{go\_id} : y, \text{name} : z).\end{array}$$

which is Boolean and asks whether the function of the gene AC04654 is known. As there is no direct link from Genbank to Go, the query has to access SwissProt in between.

The following query asks for all the gene_ids from proteins wich are involved in pathway 0052 and have as function "catalytic activity":

$$\begin{array}{rl}Q'(y) \leftarrow & \text{Kegg}(\text{pathway\_id} : \text{'0052'}, \text{protein\_id} : x),\\ & \text{SwissProt}(\text{protein\_id} : x, \text{go\_id} : z),\\ & \text{Go}(\text{go\_id} : z, \text{name} : \text{'catalytic activity'}),\\ & \text{Genbank}(\text{gene\_id} : y, \text{protein\_id} : x).\end{array}$$

□

We conclude this section, by introducing the Shortest Common Supersequence (SCS) problem which is used in Section 4 and 5. For a finite alphabet Σ, a string $s = a_1 \cdots a_n$ is a finite sequence of Σ-symbols. We denote the empty string by ε and the set of all strings by Σ^*. A string s' is a supersequence of s iff s' is of the form $w_1 a_1 w_2 a_2 \cdots w_n a_n w_{n+1}$ where each $w_i \in \Sigma^*$. A string $s = a_1 \cdots a_n$ is non-repeating when for all $i < n$, $a_i \neq a_{i+1}$.

The Shortest Common Supersequence problem (SCS) is defined as follows. Given strings $s_1, \ldots, s_n$ and a natural number K. Is there a string of length at most K that is a supersequence of every s_i? SCS is known to be NP-complete for strings over a binary alphabet [15].

3 Distributed Evaluation

In the following, a communication is the sending of a set of queries to a specific source together with the receiving of the query results. We adapt the approach of Suciu [18] to our setting in defining what constitutes an efficient distributed evaluation algorithm:

(*) In evaluating a query on a distributed database, only a constant number of messages (independent of the data at the sources) should be send and received, and the size of each message is at most logarithmic in the size of the data.

The above means that for every set of queries a fixed number of communications should suffice and that every communication transfers a constant number of tuples.[1] This constant is independent of the distributed database, but depends on the actual queries and the distributed schema. The constant will be determined by the fan-out constraints. However, in the present paper we are not interested in the actual size of this constant: only in the knowledge that a certain constant exists.

Rather than discussing general evaluation algorithms, we employ a scheme where conjunctive queries and answers to those are transmitted. In brief, the source sends out queries and builds up a local database. Here the transmitted values can depend on received values. In the following definitions, fix a distributed schema $\mathcal{S} = (\{R_1, \ldots, R_\ell\}, \Delta)$.

Definition 1. An *evaluation protocol* is a pair $P = (\overline{\mathcal{Q}}; \bar{\xi})$ where $\overline{\mathcal{Q}} := Q_1, \ldots, Q_n$ is a finite sequence of conjunctive queries such that each query Q_i is over the relational schema $\{R_k\} \cup \bigcup_{j<i} Q_j$ for some k; $\bar{\xi}$ is a finite sequence of conjunctive queries over the relational schema $\overline{\mathcal{Q}}$.

Intuitively, a protocol issues queries one at a time to a source (R_k) thereby possibly reusing results of previous queries ($\bigcup_{j \leq i} Q_j$). We refer to the latter as the local repository. Finally, the answer to every query Q_i is computed locally by evaluating the query ξ_i on the local repository. The *size* of a protocol is the sum of the sizes of the queries.

[1] We assume a reasonable binary encoding here.

Remark 1. Apart from in the examples, we assume in the following that all attributes at the various sites are disjoint. We further assume that all variables occurring in different queries are disjoint.

We denote by $\overline{\mathcal{Q}}(\mathcal{D})$ the relational database $\bigcup_{i \leq n} Q_i(\mathcal{D})$.

Definition 2. An evaluation protocol $P = (\overline{\mathcal{Q}}; \bar{\xi})$ is *bounded* if there is a natural number N, such that for every database $\mathcal{D}$ over $(\mathcal{S}, \Delta)$, $|Q_i(\mathcal{D} \cup \bigcup_{j \leq i} Q_j)| \leq N$.

Definition 3. An evaluation protocol $(\overline{\mathcal{Q}}; \bar{\xi})$ *evaluates* a sequence of conjunctive queries $\gamma_1, \ldots, \gamma_n$ iff for every database $\mathcal{D}$, $\gamma_i(\mathcal{D}) = \xi_i(\overline{\mathcal{Q}}(\mathcal{D}))$ for all $i \leq n$.

Example 3. We refer to the conjunctive queries Q and Q' of Example 2. A protocol that evaluates Q is the following: $P_1 = (Q_1, Q_2, Q_3; \xi)$ where

$$
\begin{aligned}
Q_1(\text{protein\_id} : x_1) &\leftarrow \text{Genbank}(\text{gene\_id} : \text{'AC04654'}, \text{protein\_id} : x_1) \\
Q_2(\text{go\_id} : y_1) &\leftarrow \text{SwissProt}(\text{protein\_id} : x_1, \text{go\_id} : y_1), \\
&\quad\ Q_1(\text{protein\_id} : x_1) \\
Q_3 &\leftarrow \text{Go}(\text{go\_id} : y_1, \text{name} : z_1), Q_2(\text{go\_id} : y_1) \\
\xi &\leftarrow Q_3
\end{aligned}
$$

The intuition of the above protocol is as follows:

1. first we fetch all protein_ids related to AC04654 in Genbank.
2. Next, for every such protein_id, we fetch all related go_ids.
3. Finally, we check whether the function of any of these go_ids is known.

Note that every query Q_i only uses atoms that refer to one site or to the local repository. Further, the protocol is bounded as we have the constraints Genbank : gene_id $\rightarrow$ protein_id, organism and SwissProt : protein_id $\rightarrow$ go_id, organism.

An evaluation protocol for Q' is given next. Formally, we have $P_2 = (Q_4, Q_5, Q_6, Q_7; \xi')$ with

$$
\begin{aligned}
Q_4(\text{protein\_id} : x_2) &\leftarrow \text{Kegg}(\text{pathway\_id} : \text{'0052'}, \text{protein\_id} : x_2), \\
Q_5(\text{protein\_id} : x_2, \text{go\_id} : z_2) &\leftarrow \text{SwissProt}(\text{protein\_id} : x_2, \text{go\_id} : z_2), \\
&\quad\ Q_4(\text{protein\_id} : x_2) \\
Q_6(\text{go\_id} : z_2) &\leftarrow \text{Go}(\text{go\_id} : z_2, \text{name} : \text{'catalytic activity'}), \\
&\quad\ Q_5(\text{go\_id} : z_2) \\
Q_7(\text{gene\_id} : y_2, \text{protein\_id} : x_2) &\leftarrow \text{Genbank}(\text{gene\_id} : y_2, \text{protein\_id} : x_2), \\
&\quad\ Q_4(\text{protein\_id} : x_2) \\
\xi'(\text{gene\_id} : y_2) &\leftarrow Q_5(\text{protein\_id} : x_2, \text{go\_id} : z_2), \\
&\quad\ Q_6(\text{go\_id} : z_2), \\
&\quad\ Q_7(\text{gene\_id} : y_2, \text{protein\_id} : x_2)
\end{aligned}
$$

An evaluation protocol for the sequence of queries (Q, Q') is $P_3 = (Q_1, Q_4, Q_2, Q_5, Q_7, Q_3, Q_6; \xi, \xi')$. Note that the two last evaluation protocols are bounded and that the protocol for (Q, Q') evaluates (Q, Q'). □

Note that the notion of bounded evaluation protocol corresponds to the requirements presented in (*) at the beginning of this section. Of course, queries of the form

$$Q_5(\text{protein\_id} : x_2, \text{go\_id} : z_2) \leftarrow$$
$$\text{SwissProt}(\text{protein\_id} : x_2, \text{go\_id} : z_2), Q_4(\text{protein\_id} : x_2)$$

as in the above example use atoms referring both to a distributed site and the local repository. However, as the size of the local repository will always be bounded, the local relation can be shipped together with the query to the remote site or can be hard coded in the query.

It remains to discuss how to decide that for a given query a bounded protocol exists. The previous two examples are rather simple as for every separate query only one communication is needed for every atom to determine the tuples that make this atom true. In general, there can be atoms

$$R(A_1 : c, A_2 : x_2, A_3 : x_3, A_4 : x_4), S(B_2 : x_2, B_3 : x_3, B_3 : x_4)$$

with a constant c and fan-out constraints $R : A_1 \rightarrow A_2$, $R : A_3 \rightarrow A_4$, and $S : B_2 \rightarrow B_3$. A protocol then first needs to access R to get all possible values for x_2. We refer to the latter set as the domain of x_2. Then S can be accessed to determine the domain of x_3. Finally, R should be accessed again to compute the domain of x_4. At the same time, the set of tuples that hold in R can be obtained. One final communication is then needed to determine the tuples that hold in S. In the next proposition, we show that this strategy of limiting the domain of variables suffices to check whether a query can be evaluated by a bounded protocol.

First, we introduce the following notion.

Definition 4. Given a sequence of queries $\bar{Q}$. Let $T_{\bar{Q}}$ be the set of pairs (A, x), where A is an attribute and x is a variable such that $A : x$ occurs in some atom of a query in $\bar{Q}$. Define the following sets: Bound_0 contains all pairs $(A, x) \in T_{\bar{Q}}$ where x is a constant. Further, Bound_{i+1} contains all pairs $(A, x) \in T_{\bar{Q}}$ such that

1. $(A, x) \in \text{Bound}_i$,
2. there is a $(B, x) \in \text{Bound}_i$ for some $B \neq A$; or,
3. there is an atom $R(\ldots, A_1 : x_1, \ldots, A_n : x_n, A : x, \ldots)$ such that each $(A_j, x_j) \in \text{Bound}_i$ and there is a constraint $R : \{A_1, \ldots, A_n\} \rightarrow Y$ where $A \in Y$.

Since $\text{Bound}_i \subseteq T_{\bar{Q}}$ for every i, there is an n such that $\text{Bound}_n = \text{Bound}_{n+1}$. Let Bound equal Bound_n for the smallest such n. We refer to n as the *depth* of $\bar{Q}$. We call all pairs in Bound *bounded*.

Note that the above definition induces a polynomial time algorithm to decide whether a pair is bounded.

A variable x is *local* if it only occurs in atoms that correspond to the same site and it does not occur in any of the heads.

Theorem 1. Given a sequence $\bar{Q}$ of queries over $(\mathcal{S}, \Delta)$. There is a bounded protocol P that evaluates $\bar{Q}$ iff every pair (A, x) in $T_{\bar{Q}}$ where x is not local is bounded. Moreover, the size of P is at most linear in the size of $\bar{Q}$ and $(\mathcal{S}, \Delta)$.

Proof. Suppose that every pair (A, x) in $\bar{Q}$ where x is not local is bounded. The protocol that evaluates $\bar{Q}$ proceeds by computing for every such variable its domain. In the worst case, it needs one communication for every pair in $T_{\bar{Q}}$. Then it needs to check which assignments of values to the variables makes each of the queries true. To this end, it needs to contact each site at most once. In this last step, the local variables can be evaluated. Hence, the size of the protocol is at most linear in the size of $\bar{Q}$ and $(\mathcal{S}, \Delta)$. The protocol is described more formally in the appendix.

For the other direction, suppose there is a non-local variable x such that no pair (A, x) is bounded. Let (A, x) and (A', x) be two pairs in $T_{\bar{Q}}$ occurring in two atoms L and L', respectively. Then it is easy to show by a fooling set technique from communication complexity that no protocol sending a logarithmic number of bits can check whether there is an assignment to the variables of L and L' that satisfies them both [12]. □

Corollary 1. *For a sequence of queries, it is decidable in polynomial time whether there is a bounded protocol that evaluates it.*

As we are interested in minimizing the number of different communications to the various sites, we define the following notion. Let $P = (Q_1, \ldots, Q_n; \bar{\xi})$ be an evaluation protocol. An *ordered partition of* P is an ordered sequence $1 = i_0 < \cdots < i_k = n$ of integers such that all $Q_{i_j}, \ldots, Q_{i_{j+1}-1}$ are queries over the same relational schema. The size of the partition is k.

Definition 5. The *communication size* of an evaluation protocol P, denoted by $\mathrm{cs}(P)$, is the minimal size of all its ordered partitions.

Example 4. We refer to the protocols of Example 3. The communication sizes of P_1 and P_2 are 3 and 4, respectively, while that of P_3 is 5. For the latter, the ordered partition is $\{1\}, \{4\}, \{2, 5\}, \{7\}, \{3, 6\}$. Here, the queries are to the sites Genbank, Kegg, SwissProt,Genbank, and Go, respectively. □

Definition 6. A bounded protocol is *minimal for a sequence of conjunctive queries* if there is no bounded protocol with a smaller communication size.

Proposition 1. Given a sequence of queries $\bar{Q}$. If there is a bounded protocol that evaluates $\bar{Q}$, then its communication size is always less than or equal to the sum of the sizes of the queries in $\bar{Q}$.

Proof. It suffices to note that the bounded protocol sketched in the proof of Theorem 1 has the required size. □

4 Decision problems

We define the decision problem central to the paper:

Definition 7. Given a natural number K, a distributed schema $(\mathcal{S}, \Delta)$, and, a sequence of conjunctive queries $\bar{Q}$ over $(\mathcal{S}, \Delta)$, MIN-COM is the problem to decide whether there is a bounded evaluation protocol for $\bar{Q}$ of communication size at most K.

By Proposition 1 it does not matter whether K is given in unary or binary as the size of a minimal protocol is at most linear in the size of the input. It is easy to see that MIN-COM is in NP.

Proposition 2. MIN-COM *is in* NP.

4.1 Lower bounds

It is hardly surprising that MIN-COM is in fact NP-complete. However, we prove the latter for two restricted cases. In the following, a fan-out constraint $R : X \to Y$ is *unary*, when $|X| = |Y| = 1$.

Consider the following decision problems:

1. MIN-COM$^{\text{depth } k}$ is the problem MIN-COM where in addition all fan-out constraints are unary and every input sequence of queries has depth at most k;
2. MIN-COM$_{\mathcal{S},\Delta}$ is the problem MIN-COM where in addition all fan-out constraints are unary and the queries are over the fixed distributed schema $(\mathcal{S}, \Delta)$.

The first problem gravely restricts the way in which the domain of every variable can be determined: in a constant number of steps. Intractability can then be encoded by allowing an unbounded number of relations. The second problem corresponds to the more realistic situation where the database schema is fixed in advance. In this case, intractability can be encoded by allowing arbitrarily entangled input queries. In Section 4.2, we show that when both restrictions are enforced, we get a tractable problem. Our results, hence, provide a complete picture of the worst-case complexity of the problem.

Theorem 2. *1.* MIN-COM$^{depth\ 2}$ *is* NP*-hard.*
2. MIN-COM$_{\mathcal{S},\Delta}$ *is* NP*-hard.*

Proof. (1) We use a reduction from Feedback Vertex Set (FVS) [8] which is known to be NP-complete. The problem is defined as follows. Given a directed graph $G = (V, E)$, with V a set of vertices and $E \subseteq V \times V$ a set of edges, and a natural number K. Is there a feedback vertex set of size at most K, i.e., a subset $V' \subseteq V$ such that V' contains at least one vertex from every directed cycle in G? Here, only cycles of length greater than one are considered.

Let $G = (V = \{v_1, \ldots, v_n\}, E)$ be a graph and K a natural number. Then define $\mathcal{S} = \{R_1, \ldots, R_n\}$, where each relation R_i corresponds to the node v_i. The

relation R_i has the attributes A^i, A^i_i, and for every j such that $(v_j, v_i) \in E$, an attribute A^i_j. We have the following fan-out constraints, for every i, $R_i : \{A^i\} \to \{A^i_i\}$. Let c be a constant. Then $\bar{Q}$ consists of the single query containing the following atoms: for all i,

$$R_i(A^i : c, A^i_i : v_i, A^i_{i_1} : v_{i_1}, \ldots, A^i_{i_n} : v_{i_n})$$

where $v_{i_1}, \ldots, v_{i_n}$ are all nodes for which $(v_{i_j}, v_i) \in E$. Note that the depth of $\bar{Q}$ is two. Indeed, every $A^i : c$ is of depth zero, every $A^i_i : v_i$ is of depth one and all other pairs are of depth two.

It can be argued that G has a feedback vertex set of size at most K iff there is a bounded evaluation protocol for $\bar{Q}$ of communication size at most $K + |V|$.

(2) Define SCS-NR as the problem SCS where every input string is non-repeating (cf. Section 2).

Lemma 1. *For a fixed alphabet of arity at least four,* SCS-NR *is* NP*-complete.*

Fix the alphabet $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$. We now reduce SCS-NR to MIN-COM. Let $s_1, \ldots, s_n$ be a sequence of non-repeating strings and K be a natural number. Define the binary relations σ_i with attributes A and B. For every i, we have the fan-out constraint $\sigma_i : \{A\} \to \{B\}$.

Let $s_i = s_{i1} \cdots s_{in_i}$. For $i \leq n$ and $2 \leq j \leq n_i$, let L_{ij} be the atom $s_{ij}(A : x_{ij}, B : x_{i(j+1)})$. Define L_{i1} as the atom $s_{i1}(A : c, B : x_{i2})$ for a constant c. Then define Q as the query consisting of all atoms L_{ij}.

We show that $s_1, \ldots, s_n$ has a supersequence of length at most K iff there is a bounded protocol of communication size at most K that evaluates Q.

Let s be a supersequence of length at most K. Clearly, the protocol that accesses the relations in the order induced by s is bounded and determines the domain of all variables. A query over the local repository then evaluates Q.

Conversely, let P be a protocol of communication size at most K that evaluates Q. Let $s = s_{i_1 1} \cdots s_{i_\ell \ell}$ be the order in which the different sites are addressed. As the s_i's are non-repeating, P cannot evaluate two successive $s_{ij}, s_{i(j+1)}$ with a single communication. Hence, $\ell \leq K$. As P evaluates Q and hence determines all the variables, every string s_i has to be a subsequence of s. □

4.2 A tractable case

Define MIN-COM$^{\text{depth } k}_{\mathcal{S}}$ as the problem MIN-COM where every input sequence of queries has depth at most k and the queries are over the distributed schema $\mathcal{S}$. So, $\mathcal{S}$ is given but not Δ.

Theorem 3. MIN-COM$^{depth\ k}_{\mathcal{S}}$ *is in* P.

Proof. Let $\mathcal{S} = \{R_1, \ldots, R_m\}$. We first argue that the minimal protocol is at most of communication size $m^k + m$. Indeed, following the construction in the proof of Proposition 1, the protocol first determines the domain of all variables.

As the depth of every input sequence of queries $\bar{Q}$ is k, for every pair $(A, x) \in T_{\bar{Q}}$, k communications suffice to bound the value of x in the atom it appears in. So, when executing all communication sequences of length k one after another, the domain of every variable is known. This needs m^k communications in total. Then, at every site it needs to be checked which assignments of variables make the atoms true. This needs another m communications as there are m sites. So, $m^k + m$ is an upper bound for the communication size of the minimal protocol evaluating $\bar{Q}$.

To find the minimal protocol, we only need to consider protocols of communication size at most $m^k + m$. The minimal one can be found by exploring a search tree of depth $m^k + m$ and width m. At every step there is the choice to access one of the m sites. An access to relation R_i determines as many values of variables as possible in atoms referring to R_i or when all variables are known for an atom, fetches all tuples that make that atom true. In the end, the protocol with the least communication size is taken. □

5 Heuristics

Although the algorithm described in the proof of Theorem 3 is in polynomial time, the degree of the polynomial is too high to be useful in practice. Therefore, we present in this section four heuristic algorithms to approximate MIN-COM. They are experimentally evaluated in the next section.

5.1 Greedy

The Greedy method proceeds by bounding the domains of variables. When for a certain site, only one more access is necessary to bound the domain of every variable in every atom that refers to that site, we call that site *fully determined*. We can then bound the domain of these last variables together with evaluating every such atom by one communication to the site. The latter is also the final access to that site. Therefore, the algorithm gives priority in accessing fully determined relations. If no site is fully determined, the protocol chooses to access that site which maximizes the number of variables that become bounded. This is the greedy step. We formally describe the algorithm and illustrate it by means of an example.

We introduce some terminology. Assume given a sequence of queries $\bar{Q} = Q_1, \ldots, Q_\ell$. Let $\{R_1, \ldots, R_n\}$ be the relational schema. We define the set of bound variables w.r.t. to the sequence of accesses to the different sites. Therefore, let $s \in \{1, \ldots, n\}^*$, where $s = 123$ means that we first access site R_1, then R_2 and finally R_3. Define Bound_ε as the set containing all pairs $(A, x) \in T_{\bar{Q}}$ for x a constant. Further, $\text{Bound}_{s \cdot i}$ contains Bound_s and all pairs (A, x) such that

- x is non-local,
- no $(B, x) \in \text{Bound}_{s \cdot i}$ with $B \neq A$; and

– there is an atom $R_i(\ldots, A_1 : x_1, \ldots, A_n : x_n, A : x, \ldots)$ such that each $(A_j, x_j) \in \text{Bound}_s$ and there is a constraint $R_i : \{A_1, \ldots, A_n\} \to Y$ where $A \in Y$.

A relation R is *fully determined* at step s when every pair (A, x) in every atom in $\bar{Q}$ referring to R is in Bound_s.
We describe the Greedy method. To start let $s = \varepsilon$.

1. Let j be such that R_j is fully determined at step $s \cdot j$ and R_j is unmarked. Otherwise choose j be such that $|\text{Bound}_{s \cdot j}| \geq |\text{Bound}_{s \cdot i}|$, for all $i \neq j$ and R_j is unmarked. Otherwise if all relations are marked stop.
2. We first add for every pair $(A, x) \in \text{Bound}_{s \cdot j} \setminus \text{Bound}_s$ the query Q_x that defines the set of possible values of x to the protocol: $Q_x(A_x : x) \leftarrow R(A_1 : x_1, \ldots, A_n : x_n, A : x), Q_{x_{i_1}}(A_{x_{i_1}} : x_{i_1}), \ldots, Q_{x_{i_m}}(A_{x_{i_m}} : x_{i_m})$. Here, $\{x_{i_1}, \ldots, x_{i_m}\}$ are the variables in $\{x_1, \ldots, x_n\}$. The remainder are constants.
3. If R_j is fully determined, mark R_j and add Q_{i,R_j} for every $i \leq \ell$, defined as follows. For every query Q_i and site R, let $L_1, \ldots, L_k$ be the atoms in Q_i referring to R. Let $x_1, \ldots, x_n$ be the set of non-local variables that appear in Q_i.Define the query $Q_{i,R}(A_{x_1} : x_1, \ldots, A_{x_n} : x_n) \leftarrow L_1, \ldots, L_k, Q_{x_1}(A_{x_1} : x_1), \ldots, Q_{x_n}(A_{x_n} : x_n)$. The latter query evaluates the part of every query in $\bar{Q}$ that refers to R.
4. Set s to $s \cdot j$. Go to (1).

For every $i \leq \ell$, define ξ_i as the conjunction of all $Q_{i,R}$.

Example 5. We illustrate the approach by means of an example. Consider the distributed schema $R_1(A_1, A_2, A_3, A_4)$, $R_2(A_5, A_6, A_7, A_8)$, and $R_3(A_9, A_{10}, A_{11})$, with fan-out constraints $R_1 : A_1 \to A_2, A_3$, $R_2 : A_5 \to A_6$, and $R_3 : A_9 \to A_{10}, A_{11}$. We evaluate the following two queries:

$$\begin{aligned} Q_1 \leftarrow\ & R_1(A_1 : \text{'a'}, A_2 : x_1, A_3 : x_2, A_4 : x_3), \\ & R_2(A_5 : \text{'a'}, A_6 : x_4, A_7 : x_2, A_8 : x_3), \\ & R_3(A_9 : x_4, A_{10} : x_1, A_{11} : x_3). \\ Q_2 \leftarrow\ & R_1(A_1 : \text{'b'}, A_2 : x'_1, A_3 : x'_2, A_4 : x'_3), \\ & R_2(A_5 : \text{'b'}, A_6 : x'_4, A_7 : x'_2, A_8 : x'_5), \end{aligned}$$

Denote by $\text{Bound}'_{s \cdot i}$ the set $\text{Bound}_{s \cdot i} \setminus \text{Bound}_s$. Now, $\text{Bound}'_1 = \{A_2 : x_1, A_3 : x_2, A_3 : x'_2\}$, $\text{Bound}'_2 = \{A_6 : x_4\}$, and $\text{Bound}'_3 = \emptyset$. Note that x'_1 and x'_4 are excluded as they are local. Further, none of the relations are fully determined at this point. Set $s = 1$ and add the queries

$$\begin{aligned} Q_{x_1}(A_{x_1} : x_1) &\leftarrow R_1(A_1 : \text{'a'}, A_2 : x_1) \\ Q_{x_2}(A_{x_2} : x_2) &\leftarrow R_1(A_1 : \text{'a'}, A_3 : x_2) \\ Q_{x'_2}(A_{x'_2} : x'_2) &\leftarrow R_1(A_1 : \text{'b'}, A_3 : x'_2) \end{aligned}$$

computing the domain of the variables x_1, x_2, x'_2. Then, $\text{Bound}'_{12} = \{A_6 : x_4\}$ and $\text{Bound}'_{11} = \text{Bound}'_{13} = \emptyset$. Furthermore, R_2 is not fully determined as x_3 is an unbounded non-local variable. Set $s = 12$ and add

$$Q_{x_4}(A_{x_4} : x_4) \leftarrow R_2(A_5 : \text{'a'}, A_6 : x_4)$$

Note that $\text{Bound}'_{123} = \{A_{11} : x_3\}$ and that R_3 is fully determined. Therefore, set $s = 123$, mark R_3 and add[2]

$$\begin{aligned} Q_{x_3}(A_{x_3} : x_3) &\leftarrow R_3(A_9 : x_4, A_{11} : x_3), Q_{x_4}(A_{x_4} : x_4) \\ Q_{1,R_3}(x_1, x_3, x_4) &\leftarrow R_3(A_9 : x_4, A_{10} : x_1, A_{11} : x_3), Q_{x_1}(A_{x_1} : x_1), \\ &\qquad Q_{x_3}(A_{x_3} : x_3), Q_{x_4}(A_{x_4} : x_4) \end{aligned}$$

At this point, all non-local variables are bounded and thus all sites are fully determined. Now set $s = 1231$, mark R_1, and add

$$\begin{aligned} Q_{1,R_1}(x_1, x_2, x_3) &\leftarrow R_1(A_1 : \text{'a'}, A_2 : x_1, A_3 : x_2, A_4 : x_3), \\ &\qquad Q_{x_1}(A_{x_1} : x_1), Q_{x_2}(A_{x_2} : x_2), Q_{x_3}(A_{x_3} : x_3) \\ Q_{2,R_1}(x'_2) &\leftarrow R_1(A_1 : \text{'b'}, A_2 : x'_1, A_3 : x'_2, A_4 : x'_3), Q_{x'_2}(A_{x'_2} : x'_2) \end{aligned}$$

Next, set $s = 12312$, mark R_2, and add

$$\begin{aligned} Q_{1,R_2}(x_4, x_2, x_3) &\leftarrow R_2(A_5 : \text{'a'}, A_6 : x_4, A_7 : x_2, A_8 : x_3), Q_{x_4}(A_{x_4} : x_4), \\ &\qquad , Q_{x_2}(A_{x_2} : x_2), Q_{x_3}(A_{x_3} : x_3) \\ Q_{2,R_2}(x'_2) &\leftarrow R_2(A_5 : \text{'b'}, A_6 : x'_4, A_7 : x'_2, A_8 : x'_5), Q_{x'_2}(A_{x'_2} : x'_2) \end{aligned}$$

Finally, add to ξ

$$\begin{aligned} \xi_1 &\leftarrow Q_{1,R_1}(x_1, x_2, x_3), Q_{1,R_2}(x_2, x_3), Q_{1,R_3}(x_1, x_3, x_4) \\ \xi_2 &\leftarrow Q_{2,R_1}(x'_2), Q_{2,R_2}(x'_2) \end{aligned}$$

Note that the constructed protocol is not minimal. The minimal protocol can be constructed from the sequence 2312. □

5.2 SCS Majority-Merge (MM)

We now compute for every separate query in $\bar{Q}$ a minimal protocol by exhaustive search. All the obtained minimal protocols for the separate queries are then combined in an overall protocol by using the Majority-Merge algorithm [7, 10] which is an approximation of SCS. The latter is illustrated in Example 6.

First, we explain how a minimal protocol is computed for every query. We consider all possible sequences of accesses to the sites. At every access we determine the domains of as many variables as possible. Whenever the domain of every variable in an atom is determined, that atom is evaluated. For simplicity, in the sequel, we only talk about the order in which we access the sites and do not give the concrete queries. It should be understood that they follow the strategy outlined above. The latter brute-force approach is feasible as the size of each separate query is expected to be small, say consisting of around 10 atoms.

Example 6. Assume we have three queries Q_1, Q_2 and Q_3 whose respective minimal protocols access the sites in the following order: $R_1R_2R_3R_2R_1$, $R_1R_3R_1R_2$ and $R_2R_3R_1$. The next step is to find an overall protocol which is a supersequence of every single protocol. The Majority-Merge algorithm iteratively adds

[2] To keep queries readable we omit the attributes in the heads of each Q_{i,R_j}.

the symbol that occurs the most among the leftmost symbols of the remaining sequences and removes it from those sequences. The following overview shows the respective iterations for the given sequences:

1	2	3	4	5	6
R_1	R_2	R_3		R_2	R_1
R_1		R_3	R_1	R_2	
	R_2	R_3	R_1		

The obtained supersequence then is $R_1R_2R_3R_1R_2R_1$.

Improvement. As explained above, every generated protocol has two kind of queries: those that get the domain of bounded variables and those that evaluate atoms. At a certain point, a protocol only contains queries of the second kind. We refer to this as phase two. Clearly, the order of the calls in the second phase is irrelevant. This means that every permutation of the sites in the second phase leads to another minimal protocol. We exploit this fact by aligning only the first phases of the protocols and then checking which sites still have to be added to the protocol. We denote this heuristic method with iMM.

Example 7. We take the same queries as in Example 6. Suppose the first phases for the three queries are $R_1R_2R_3R_2$, R_1R_3 and R_2R_3. The Majority-Merge algorithm returns the sequence $R_1R_2R_3R_2$. The next step is to check for every query which of the relations in the second phase still have to be added to the sequence. For the first query R_1 has to be added and the sequence now becomes $R_1R_2R_3R_2R_1$. For the second and third query nothing has to be added, as the sequence formed by their first phase and a permutation of their second phase is a subsequence of the overall protocol.We, hence, obtain a shorter overall protocol. □

5.3 Pairwise SCS (PSCS)

Even excluding permutations of calls in the second phase, some queries have more than one minimal protocol. The choice of which minimal protocol to use to construct the overall protocol can therefore strongly affect the overall protocol. In the PSCS-approach we consider all minimal protocols for every separate query (rather than just one), but construct the overall protocol by pairwise alignment as it is known that the SCS problem for two sequences is solvable in polynomial time [21].

We outline the PSCS algorithm:

1. Compute for every separate query the set of all minimal protocols by exhaustive search. For a query Q, denote by S_Q the set of sequences corresponding to the first phases of the minimal protocols.
2. Take two arbitrary queries Q_1 and Q_2 in $\bar{Q}$. Compute for every pair of sequences in $S_{Q_1} \times S_{Q_2}$ its shortest common supersequence. Let s be the shortest among all of these.

3. For every remaining query Q, compute the shortest common supersequence of s and each $s_Q \in S_Q$. Set s to be the shortest among them.
4. Add second phases to s as long as necessary like in the iMM-approach.

6 Experiments

Next, we experimentally validate our four algorithms. We randomly generate 1000 queries of varying length. To be precise, the number of atoms for each query is drawn from a Poisson-distribution with average size 10. Relations for atoms are randomly selected. Variables and data values are randomly assigned to the attributes of these relations. The experiments were performed on a Pentium IV (3.0 GHz) architecture with 1 GB of internal memory running under Linux 2.6. All programs are written in Java.

We considered two kinds of schemas: (1) randomly generated schemas with 10 relations and random fan-out constraints allowing for queries of at least depth five; (2) a fixed biologically motivated schema given in [20] created by examining popular life science web sites.

Figures 1 and 2 present a box plot of the sizes of the protocols produced by the four algorithms over random schemas and the biological schema, respectively. In brief, the lower and upper ends of the box indicate the 25th and 75th percentiles, respectively, while the line inside the box indicates the 50th percentile. The top and bottom lines of the tails indicate the 10th and 90th percentile (cf., e.g., [19].). A circle indicates an outlier. The box plots visualize data from 20 and 10 experiments, respectively. It is immediate that iMM provides a serious improvement over MM. In the case of random schemas, Figure 1 already indicates that PSCS performs better than the other methods. Further, a T-test on the data generated by the experiments establishes that the average length of protocols generated by PSCS is significantly smaller than those generated by the other methods. In the case of the biological schema, the visualization in Figure 2 alone already shows that the Greedy method outperforms all the others. The reason is that the complexity of the structure of the fan-out constraints for the biological schema is far less complicated than those of the randomly generated schemas. It appears that the Greedy method has a better performance in such a situation. Furthermore, PSCS performs better than iMM.

Figure 3 shows that for small numbers of queries, our three heuristics Greedy, iMM, and PSCS, generate a protocol whose length is close to the length of the optimal protocol (computed by exhaustive search). For larger numbers of queries it was not possible to obtain a solution by brute-force search.

Finally, we compare running times in Figure 4. While the SCS-based methods iMM and PSCS are very fast, the Greedy method is several orders of magnitude slower. The bottleneck of the algorithm is in the computation of the sets $\text{Bound}_{s \cdot j}$ for which every atom in every query has to be accessed in every iteration of the algorithm.

7 Conclusion

We proved MIN-COM to be an intractable problem (even under severe restrictions) and provided four heuristics. When to use which heuristic depends on the setting.

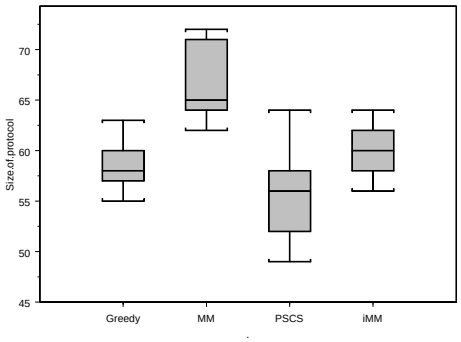

Fig. 1. Box plot of protocol size for 20 experiments with 1000 queries over a random schema.

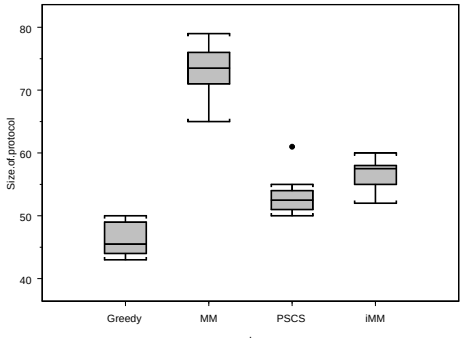

Fig. 2. Box plot of protocol size for 10 experiments with 1000 queries over the biological schema.

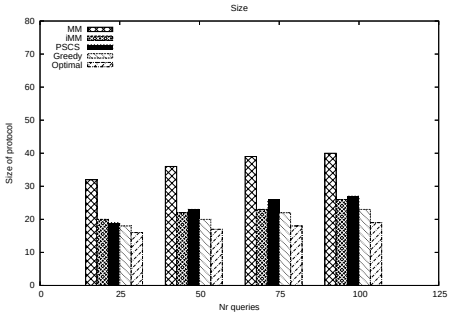

Fig. 3. Comparison of protocol size with optimal solution for a small number of queries.

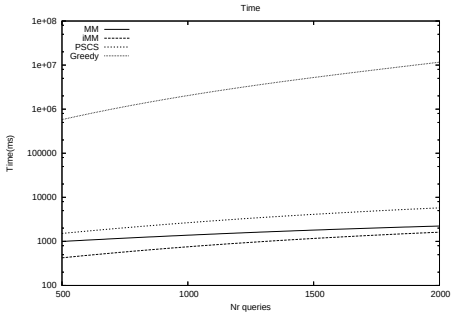

Fig. 4. Average logarithmic time.

Our experiments show that in a setting with a schema with a low complexity structure of fan-out constraints, as is the case in our biological scenario, the Greedy method performs best. In a random scenario PSCS outperforms the other methods. The latter method has the additional advantage that it is much faster than the Greedy method: 2.7 hours (Greedy) versus 10 seconds (PSCS) for 2000 random queries. The main drawback of the present approach is that only the number of accesses to sites is minimized, the overall amount of transmitted data remains the same. In future work we plan to address that issue.

Appendix

Proof of Theorem 1(continued) We formally describe the protocol. For $i = 1, \ldots, n$, add the following conjunctive queries to the protocol. For every pair $(A, x) \in \text{Bound}_i \setminus \text{Bound}_{i-1}$, that has been added to Bound_i

- by the second rule in Definition 4, add $Q_{A:x}(A : x) \leftarrow Q_{B:x}(B : x)$; and,
- by the third rule in Definition 4, add $Q_{A:x}(A : x) \leftarrow R(A_1 : x_1, \ldots, A_n : x_n, A : x), Q_{A_{i_1}:x_{i_1}}(A_{i_1} : x_{i_1}), \ldots, Q_{A_{i_m}:x_{i_m}}(A_{i_m} : x_{i_m})$. Here, $\{x_{i_1}, \ldots, x_{i_m}\}$ are the variables in $\{x_1, \ldots, x_n\}$. The remainder are constants.

Up to now, the protocol computes a set of possible values for all bound variables. It remains to test which variable assignments hold at the databases. Therefore, for every query Q_i and site R, let $L_1, \ldots, L_k$ be the atoms in Q_i referring to R. Let $x_1, \ldots, x_n$ be the set of non-local variables that appear in Q_i with the attributes $A_1, \ldots, A_n$, respectively. Define the query $Q_{i,R}(x_1, \ldots, x_n) \leftarrow L_1, \ldots, L_k, Q_{A_1:x_1}(A_1 : x_1), \ldots, Q_{A_n:x_n}(A_n : x_n)$. Add all these to the protocol. It remains to define all ξ. For every query Q_i, define ξ_i as the conjunction of all $Q_{i.R}$. □

Proof of Proposition 2. Given $(K, \overline{\mathcal{Q}}, \mathcal{S}, \Delta)$, we simply guess a protocol P of at most linear size, check whether it is bounded and whether it is of communication size at most K. This can be done in polynomial time. We then only need to verify whether that P effectively evaluates $\overline{\mathcal{Q}}$. The latter reduces to testing equivalence of conjunctive queries which is known to be in NP [6] by guessing of homomorphisms. These guesses can be combined with the guessing of the protocol. The latter verification step can be combined with the first verification step. □

Proof of Theorem 2(1) (continued). Note that from the construction of $\bar{Q}$, a minimal bounded protocol accesses every relation R_i at least once and at most twice: once to apply the fan-out constraint to bound the possible values for v_i, and once to evaluate the atom $R_i(A^i : c, A^i_i : v_i, A^i_{i_1} : v_{i_1}, \ldots, A^i_{i_n} : v_{i_n})$ when all variables $v_i, v_{i_1}, \ldots, v_{i_n}$ are determined. Call relations that are accessed twice *expensive*, and the others *cheap*. Now, every minimal protocol can be rewritten into one that accesses first all expensive atoms, then all cheap ones, and finally all expensive atoms again. Therefore, minimizing the communication size of the protocol reduces to minimizing the number of expensive atoms. By construction, $A^i_j : v_j$ is added to the atom R_i when there is an edge from v_j to v_i in G. This means that in order for R_i to be cheap, all values of variables corresponding to incoming edges should be known when evaluating R_i. So, for every directed cycle there has to be one node v_i such that R_i is expensive. Therefore, the set $V' = \{v_{j_1}, \ldots, v_{j_\ell}\}$, where $\{R_{j_1}, \ldots, R_{j_\ell}\}$ is the set of expensive atoms, forms a feedback vertex set. So, if there is bounded protocol of size at most $K + |V|$, where K is the number of expensive atoms, there is a feedback vertex set of size K.

Conversely, assume that $V' = \{v_{j_1}, \ldots, v_{j_\ell}\}$ is a feedback vertex set of size at most K. We only describe the order in which sites are accessed:

- Access the relations $R_{j_1}, \ldots, R_{j_\ell}$ to determine the values of the variables $v_{j_1}, \ldots, v_{j_\ell}$. For every, such v, we have a query Q_v.
- Remove from G all nodes in V'. Select a node v with in-degree zero. Note that this is possible as the resulting graph is acyclic. As v has in-degree zero in the resulting graph, this means that all the values of all variables v' for which (v', v) in the original graph G, are known. Hence, we can check with a single communication which tuples make R_v true using the local repository. Finally, remove v and repeat until no nodes are left.
- Access the relations $R_{j_1}, \ldots, R_{j_\ell}$ to determine which tuples make them true.
- Check whether there is an assignment to the variables that makes all atoms true. The latter is a query over the local repository and does not need any access to a remote site.

Note that the above protocol evaluates $\bar{Q}$, is bounded and has communication size $K + |V|$.

□

Proof of Lemma 1. We use a reduction from SCS for strings over a binary alphabet $\{\alpha, \beta\}$. Let S be a set of strings $\{s_1, \ldots, s_n\}$, and K a natural number. Then define Σ' as $\{\alpha, \beta, \alpha', \beta'\}$ and S' as the set of strings obtained from S by replacing in every s every occurrence of α and β by $\alpha\alpha'$ and $\beta\beta'$, respectively. Note that this construction ensures that the strings in S' are non repeating.

We show that S has a supersequence s of length at most K iff S' has a supersequence s' of length at most $2K$.

Clearly, if s is supersequence for S of length at most K, then s' obtained from s by applying the above transformation, is a super sequence of size at most $2K$.

Conversely, suppose s' is a supersequence for S' of length at most $2K$. Let s be obtained from s' by eliminating all symbols α' and β'. Clearly, s is a supersequence for S. It remains to argue that the length of s is at most K. This is definitely the case if α and β occur the same number of times or less in s' as their corresponding symbols α' and β'. Therefore, suppose α occurs more often than α'. This means that their is an occurence of α followed by another occurence of α before the next occurrence of α' or not followed by an occurence of α' at all. But then this occurrence of α is superfluous because in the strings in S' every occurrence of α is directly followed by an occurrence of α'. We can hence keep on deleting superfluous occurrences. As a result the length of s' is at most K. □

- PubMed(pubmed_id,abstract,article)
  - PubMed: pubmed_id → abstract,article
- Kegg(EC_number,pathway_id,protein_id)
  - Kegg : EC_number → pathway_id, protein_id
  - Kegg : pathway_id → protein_id
  - Kegg : protein_id → EC_number
- OMIM(omim_id,gene_id,map_locus,pubmed_id)
  - OMIM: omim_id → gene_id,map_locus,pubmed_id
  - OMIM: map_locus → omim_id,gene_id,pubmed_id
  - OMIM: gene_id → omim_id,map_locus,pubmed_id
- GO(go_id,name,definition,cellular_component)
  - GO : go_id → name,definition,cellular_component
- SwissProt(protein_id,protein_name,EC_number,go_id,omim_id,gene_id,pubmed_id)
  - SwissProt : protein_id → protein_name,EC_number,go_id,omim_id,gene_id,pubmed_id
  - SwissProt : protein_name → protein_id,EC_number,go_id,omim_id,gene_id,pubmed_id
  - SwissProt : omim_id → pubmed_id
  - SwissProt : gene_id → pubmed_id,protein_id
  - SwissProt : EC_number → protein_id
- Genbank(gene_id,gene_name,pubmed_id,feature_id)
  - Genbank : gene_id → gene_name,pubmed_id,feature_id
  - Genbank : gene_name → gene_id,pubmed_id,feature_id
  - Genbank : feature_id → gene_id
- GenbankFT(feature_id,organism,go_id,omim_id,Entrez_protein_id)
  - GenbankFT: feature_id → organism,go_id,omim_id,Entrez_protein_id
  - GenbankFT: Entrez_protein_id → feature_id
- SageGenie(gene_id,tag)
  - SageGenie : gene_id → tag
  - Genbank : tag → gene_id
- UniGene(cluster_id,gene_id,Entrez_protein_id)
  - UniGene : cluster_id → gene_id,Entrez_protein_id
  - UniGene : Entrez_protein_id → cluster_id
- HomoloGene(homolgene_id,Entrez_protein_id,blast2_genesequence)
  - HomoloGene : homolgene_id → Entrez_protein_id,blast2_genesequence
- Mesh(gene_id,definition)
  - Mesh : gene_id → definition
- EntrezProtein(Entrez_protein_id,protein_name,cluster_id)
  - EntrezProtein : Entrez_protein_id → protein_name,,cluster_id
  - EntrezProtein : protein_name → Entrez_protein_id,cluster_id
  - EntrezProtein : cluster_id → Entrez_protein_id
- EntrezGene(gene_id,map_locus)
  - EntrezGene : gene_id → map_locus
  - EntrezGene : map_locus → gene_id

Fig. 5. The biological schema

References

1. M. Altinel and M. J. Franklin. Efficient filtering of XML documents for selective dissemination of information. In *Proc. of the 26th International Conference on Very Large Data Bases (VLDB 2000)*, pages 53–64. Morgan Kaufmann, 2000.
2. M. Ashburner et al. Gene Ontology: tool for the unification of biology. *Nature Genetics*, 25(1):25–29, 2000.
3. A. Bairoch and R. Apweiler. The SWISS-PROT protein sequence data bank and its new supplement TREMBL. *Nucleic Acids Research*, 24(1):21–25, 1996.
4. P. A. Bernstein, N. Goodman, E. Wong, C. L. Reeve, and Jr. J. B. Rothnie. Query processing in a system for distributed databases (SDD-1). *ACM Transactions on Database Systems*, 6(4):602–625, 1981.
5. H.S. Bilofsky et al. The GenBank Genetic Sequence Databank. *Nucleic Acids Research*, 14:1–4, 1986.
6. A. Chandra and P. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings 9th ACM Symposium on Theory of Computing (STOC 1977)*, pages 77–90. ACM Press, 1977.
7. D. E. Foulser, M. Li, and Q. Yang. Theory and algorithms for plan merging. *Artificial Intelligence*, 57(2-3):143–181, 1992.
8. M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, 1979.
9. K. Hokamp and K. Wolfe. What's new in the library? What's new in GenBank? Let PubCrawler tell you. *Trends in Genetics*, 15(11):471–472, 1999.
10. T. Jiang and M. Li. On the approximation of shortest common supersequences and longest common subsequences. *SIAM Journal on Computing*, 24(5):1122–1139, 1995.
11. M. Kanehisa and S. Goto. KEGG: kyoto encyclopedia of genes and genomes. *Nucleic Acids Research*, 28(1):27–30, 2000.
12. E. Kushilevitz and N. Nisan. *Communication complexity*. Cambridge University Press, 1997.
13. Z. Lacroix and T. Critchlow. *Bioinformatics: Managing Scientific Data*. Morgan Kaufmann, 2003.
14. H. Lu, B. Ooi, and C. Goh. On global multidatabase query optimization. *SIGMOD Record*, 21(4):6–11, 1992.
15. K.J. Raeiha and E. Ukkonen. Shortest common supersequence problem over binary alphabet is NP-complete. *Theoretical Computer Science*, 16(2):187–198, 1981.
16. P. Roy, S. Seshadri, S. Sudarshan, and S. Bhobe. Efficient and extensible algorithms for multi query optimization. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data (SIGMOD 2000)*, pages 249–260. ACM Press, 2000.
17. A. P. Sheth and J. A. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3):183–236, 1990.
18. D. Suciu. Distributed query evaluation on semistructured data. *ACM Transactions on Database Systems*, 27(1):1–62, 2002.
19. P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to data mining*. Addison-Wesley, 2005.
20. D. Van de Craen. Biologically motivated schema. `http://alpha.uhasselt.be/~lucp1631/files/biodbschema.pdf`

21. R. A. Wagner and M. J. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21(1):168–173, 1974.
22. C. Wang and M. Chen. On the complexity of distributed query optimization. *IEEE Transactions on Knowledge and Data Engineering*, 8(4):650–662, 1996.