Evaluating Very Large Datalog Queries on Social Networks

Royi Ronen and Oded Shmueli[*]
Technion - Israel Institute of Technology
{ronenr,oshmu}@cs.technion.ac.il

ABSTRACT
We consider a near future scenario in which users of a Web 2.0 application, such as a social network, contribute to the application not only data, but also rules which automatically query, utilize and create the data. For example, a user of a social network can define rules that automatically manage the user's friends list, the sending of various announcements, filtering of messages and more.

We examine the probable case of automated addition of connections by a participant. The connections to be added are defined using a query, associated to each participant. For this, we introduce and study the *Query Network* model, a graph-based model in which every node models a network participant and is associated with a Datalog rule. The union of all these individual user rules constitutes a very large, recursive, Datalog program whose size is of the order of magnitude of the size of the data being queried (data whose size in a social network can easily exceed 1TB). This greatly differs from the traditional assumption that queries are small and data are large. In particular, traditional optimizers will be hard pressed to handle such queries. This is the case even if queries are 'translated' to SQL (using views) and their union is transformed to a very large SQL query.

We have designed, built and experimented with evaluation algorithms for such query networks. Experiments with both synthetic and real datasets demonstrate the usefulness and high effectiveness of our methods. Extensions to the model are proposed, their implementation and testing are the subject of on-going work.

1. INTRODUCTION
Web 2.0 is a general term for internet applications which interconnect their end users either by providing a connectivity platform or by enabling users to share contents with each other (or both). Some examples for Web 2.0 applications are blog-hosting sites, wikis, video-sharing sites, dating sites, instant messaging applications and social networks. For example, Wikipedia [30] is an encyclopedia whose entries are authored, edited and maintained by the users, as opposed to, e.g., the Britannica online encyclopedia [6] which simply provides users with contents.

A highly successful type of Web 2.0 application is the Social Network. Some examples of social networks are Facebook [15], MySpace [23] and LinkedIn [20]. Social networking applications also exist within corporations, universities and other organizations. Typically, a participant in a social network is associated with some information (such as name, photograph, interests), and with a list of connections to other participants.

Beside being a useful platform for social interactions, social networks are increasingly being used as a tool for business development and management. LinkedIn [20], for example, is a social network dedicated to professional networking. The usefulness and popularity of social networks results in increasing amounts of data that are available to their users. Such amounts of data are hard and expensive to manage manually. Indeed, in Facebook, users can take advantage of a simple query language [16] in order to process, mostly, their own and their immediate friends' data. However, the main social networking feature of managing connections between participants is still managed manually.

In addition to being tools to manage increasing volumes of data using queries, we believe that automatic data management features for social networks will be required in order to (conceptually) provide interaction between users even when they are not online. Ultimately, these features will evolve to become *agents* which represent, and act on behalf of, the participants in the network.

In this paper, we consider a near future scenario in which participants of a social network manage their data automatically. In particular, we examine the probable case in which participants define, in a form of a query, with whom they would like to be connected.

1.1 Motivational Example
Consider the social network illustrated in Figure 1. Seven participants are shown with their current connections in the network (the connections are represented by directed edges,

[*]R. Ronen was funded in part by ISF 1104/05.
O. Shmueli was funded by ISF grant 1104/05.

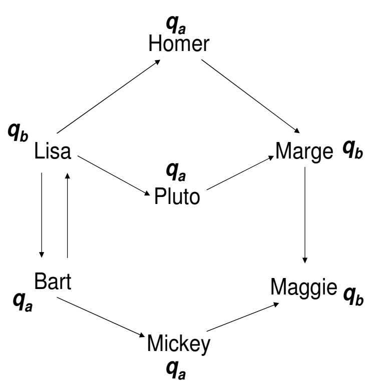

Figure 1: The network for the example in Section 1.1.

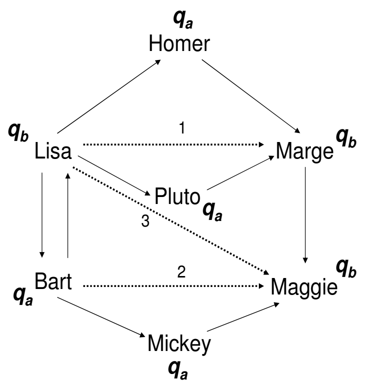

Figure 2: The network after adding connections.

an issue discussed in Section 1.2). The participants specify policies which define which connections they would like to add to themselves. Lisa would like to connect to every participant who is a friend of two distinct friends of hers. Bart, on the other hand, would like to connect to every participant who is connected to him through two distinct paths, such that one path is of length 2 edges and the other - of length 3. These two definition of policies are formally given later on as queries q_a (Lisa) and q_b (Bart) in Section 2, and are illustrated graphically in Figure 4. However, the formal definitions are not needed in the context of this general example. The rest of the participants are associated with queries as shown in Figure 1.

Lisa is not connected to Marge. However, Lisa is connected to Homer and to Pluto, who are both connected to Marge. According to Lisa's query, we add the edge *(Lisa, Marge)* to the network. In Figure 2, this edge is marked by 1. Based on this added edge, Maggie became a participant to whom Bart is connected through two paths. A two-edge path through Mickey, and a three-edge path through Lisa and Marge. We therefore add the edge *(Bart, Maggie)* to the network (marked 2 in Figure 2). Note that the addition of edge 2 is done based on edge 1 and on original network edges. The addition of edge 2 renders Maggie, who is not connected to Lisa, as a participant who is connected to two distinct friends of Lisa. The edge *(Lisa, Maggie)*, marked 3 in Figure 2 is therefore added. Note that edge 3 is added based on original edges and edge 2.

The addition of edges in this example demonstrates the recursive nature of adding edges to the network. Lisa's query was evaluated and edge 1 was added. Based on this addition (but not based on it only), another evaluation of Lisa's query resulted in the addition of edge 3. As we will shortly see, we model this recursion using the Datalog formalism.

1.2 Model

We use a graph-based formalism to model the network. Nodes model the network participants, and edges represent the links between participants. Most social networks today allow only reciprocal friends (a situation where if a lists b as a friend, then b also lists a). Nevertheless, we model connections with directed edges since directed edges are capable of modeling both networks with reciprocal friends and networks with non-reciprocal friends. In addition, the directed graph makes the definition of queries and their semantics more succinct.

Also, we believe that the fast development of social networks and their proliferation to business and organizational cultures will result in new types of connections whose modeling may require directed edges. For example, connections between fans and a rock star or between pupils and their teacher.

Every node, say n, has a query associated with it. This query defines the nodes that the participant corresponding to n would like to add to her friends' list. As in most social interactions, the query will make use of existing connections in order to create new ones.

Conjunctive Queries. Our query formalization is that of a *Conjunctive Query* (CQ) [29]. A CQ has a body, built of atoms, which are predicates with parameters [29]. A parameter can be either a variable or a constant. Each CQ has a head which is an atom. The predicates are relation names. Intuitively, a CQ adds to the relation corresponding to its head atom all tuples that correspond to a satisfying substitution of the query variables. We only allow the use of CQs that are *safe* and without negation. Refer to [29] for formal definitions of safety and CQs semantics.

The choice of the CQ model is motivated by the following considerations. The CQ is a formalization that has been successfully used in analyzing many query languages (for both structured and unstructured data) such as SQL [10], XPath [32] (and therefore XQuery [33]), XSLT [17], SQL4X [11], Elog [8] and XPath^L [26], [27]. CQs are also used to abstract SPARQL queries for Semantic Web data (RDF) [9] and XPath-inspired queries on DAGs [25]. The CQ formalization is extensively studied and well understood. Both theoretical and practical issues, such as optimization methods, have been addressed [29], [1], [24], [7].

In addition, experience shows that the CQ formalism can be successfully extended to handle data types other than relational (e.g., [11], [26], [27], [9]), a crucial characteristic in the dynamic, heterogeneous web environment.

Query Network and Datalog. A Datalog query is a generalization of a CQ [29], [1]. A Datalog query (also called a *program*) is a set of rules, each of which is essentially a CQ. The result of one rule can be used by another rule in order to produce a new result and so on. A Datalog program can be recursive, which means that a predicate can be defined in terms of itself, either within the same rule or indirectly. The initial relations constitute the *EDB* (for Extensional Database) and the derived relations form the *IDB* (for Intensional Database).

In the query network model, we look at the large collection of CQs associated with the network nodes as a Datalog program. We employ a least-fixpoint bottom-up semantics [29] for the Datalog program composed of all the CQs in the network (Section 2 provides formal definitions).

Recently, many Datalog-based languages have been developed for a variety of systems in the fields of networking and distributed systems, computer games, machine learning and robotics, compilers, security protocols and information extraction, as reported in the Claremont report on Database Research [3]. According to this report, the use of declarative languages in these applications has been successful in providing an order-of-magnitude reduction in the size of code, in comparison with other solutions. The report also regards declarative languages as an important step in a suggested prospective development of data management, from a storage service to a programming paradigm.

We believe that this development, combined with the enormous amounts of data and data types currently available to Web 2.0 users (and in other scenarios as well) will inevitably result in a growing need to process large queries. This greatly differs from the prevailing assumption in query processing - that queries are small and data are large. Traditional compilers, optimizers and evaluation techniques are not designed to handle a query whose size is in the order of magnitude of the data being queried. In particular, rewriting techniques which may increase the query size exponentially (e.g., the Magic Sets optimization for Datalog programs [5]) or add many rules to the program (e.g., the Counting Method [24]) are not applicable in this case.

Motivated by the automated Social Network scenario, we present in this paper algorithms for evaluating large Datalog queries on Social Networks.

1.3 Contributions
The main contributions of this paper are the following.

Model and Algorithms. The query network model and its motivation are introduced. Three algorithms for evaluating very large Datalog queries over a query network are presented. The *Basic* evaluation algorithm is a simple evaluation algorithm, it is used as a baseline for comparison. The *Backward-Radius Triggering* algorithm reduces the number of query evaluations by triggering only the evaluation of queries whose results may have been affected by edges added to the network. The *Divide and Conquer* evaluation algorithm partitions the network, evaluates each partition separately and merges the results. It has a large potential for parallelism.

Implementation and Experimentation. Implementation of the algorithms is presented. Experimentation with synthetically generated datasets, as well as datasets derived from the DBLP [12] database are presented. DBLP was chosen in order to capture patterns of social activity as reflected in the collaboration between authors of publications recorded in DBLP. The results show that the Backward-Radius Triggering and the Divide and Conquer evaluation algorithms perform significantly better than the Basic evaluation algorithm. Among the two, our results clearly show that the benefit of using DAC is correlative to the ratio between the sizes of the IDB and the EDB.

Extensions. Extensions to the model are proposed.

2. PRELIMINARIES
We shall make use of the following definitions.

A Query Network. A *Query Network* is a directed graph (N, F^0) where N (for *Nodes*) is a set of participants. F^0 (for *Friends*) is a set of directed edges between pairs of distinct elements of N. $F^0 \subseteq (N \times N) \setminus \{(n_0, n_0)$ s.t. $n_0 \in N\}$. Every node $n \in N$ has a query associated with it. The query defines edges of the form $(n, \cdot)$, which are edges such that n would like to add to the initial set of edges, F^0, as defined next.

A Query. A query associated with a network node n, $q(n)$, is a Datalog rule of the following form. The rule's head is $F^+(n, X)$, where X is a variable. F^+ is an IDB relation which will contain the additions to F^0, which is an EDB relation. We define another relation, F, as $F = F^+ \cup F^0$. The body of the rule is composed of predicates corresponding to the relation F and the inequality predicate. We require that one of the predicates be of the form $F(n, Y)$, and that X, the variable in the rule head, appear in one of the F body predicates. The latter requirement is added for *safety* (see [29]). Further requirements follow the example.

Example. The following query adds to F tuples of the form (n, X) where X is a friend of two distinct friends of n:

$$F^+(n,X) \leftarrow F(n,Y), F(Y,X), F(n,Z), F(Z,X), Y \neq Z, X \neq n, Y \neq n, Z \neq n.$$

We, however, henceforth assume that unless otherwise specified, each two distinct variables imply an inequality predicate between them. We further assume that none of the variables is equal to n. Therefore, the query is abbreviated to: $F^+(n,X) \leftarrow F(n,Y), F(Y,X), F(n,Z), F(Z,X).$ ■

For a query $q(n)$ we also require that n is the only constant in the query (that is, all the other arguments are variables).

Query Graph. Let the *Query Graph* of a query $q(n)$ be the graph whose nodes are the variables and n, the single constant in the query $q(n)$, and in which a directed edge ex-

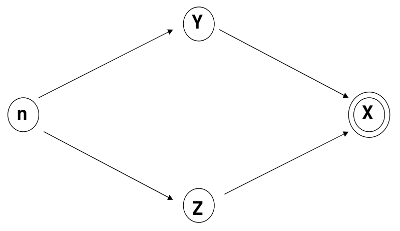

Figure 3: The query graph corresponding to the query $F(n, X) \leftarrow F(n, Y), F(Y, X), F(n, Z), F(Z, X)$.

ists between two variables X and Y (respectively, a constant n and an argument Z) if a predicate $F(X, Y)$ (respectively, $F(n, Z)$) occurs in the query body. In the graphical representation of the graph, the variable in the head of the rule appears in a double circle.

Example. The query graph corresponding to the query in the previous example is shown in Figure 3 ■.

Consider the predicate of the form $F(n, Y)$ which we require in every query body. This predicate contributes to the query graph a node which represents n. We require that every query in the query network be such that the nodes in its query graph are all reachable from the node representing n.

Radius. The *Radius* of a query q is the number of edges in the longest path, that never traverses a node more than once, in the query graph corresponding to q. Naturally, we assume that the radius of any query is very small relative to the size of the network.

Backward Radius (bradius). Intuitively, the *Backward Radius*, or *bradius*, of a node n is the maximal distance from another node m such that the query $q(m)$ can 'sense' the edges whose source is n. Formally, we define, $B(n, k) = \{m \in N| \textit{ there exists a path of length } k \textit{ from } m \textit{ to } n\}$, and $L(n, k) = \{b \mid b \in B(n, k) \textit{ and the radius of } q(b) \textit{ is at least } k\}$. The *bradius* of node n is the maximal k such that $L(n, k) \neq \varnothing$. Note that the bradius of any node is bounded by the maximal radius in the network.

Example. Figure 4 shows a small example of a query network. Each node has an id (a number) and is associated with one query, either q_a, q_b or q_c, whose query graphs also appear in Figure 4. We assume that the head of the query is always $F(n, X)$. The radius of node 2 is 3. The radius of node 4 is 2. The bradius of node 8 is 3. The bradius of node 2 is 1. ■

Single Evaluation of a Node. Consider node 4 in the network illustrated in Figure 4. Evaluating once the query associated with this node, q_b, inserts the edge $(4, 8)$ to F. We call the process of evaluating a node's query on a given network and subsequently adding a (possibly empty) set of nodes to F, an *evaluation of a node.*

Exhaustive Evaluation of a Node. Consider node 2 in the network after the edge $(4, 8)$ has been added to the

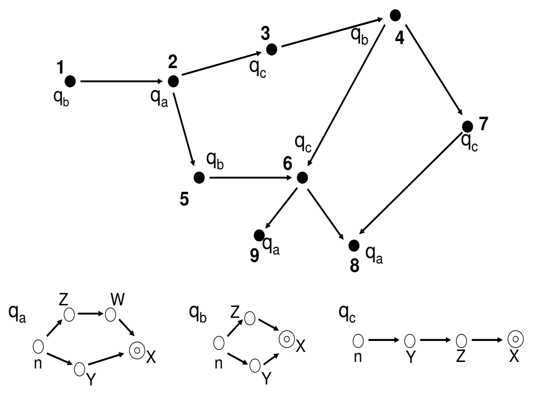

Figure 4: Query Network Example.

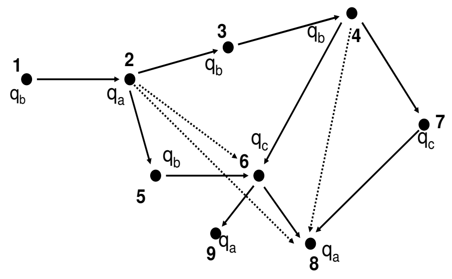

Figure 5: The network, after the additions.

original network. After a single evaluation of this node, $(2, 6)$ is in F. Since $(2, 6)$ is in F, another single evaluation of node 2 will result in the addition of $(2, 8)$ to F. Note that this is different than a single evaluation which adds two edges to F, since the addition of $(2, 8)$ is done based on the edge (2,6) that is added in the first evaluation.

The process of evaluating a single query, and this query only, repeatedly and until no edges can be added to F, is called *an exhaustive evaluation of a node.* The network after evaluation is presented in Figure 5. The new edges are dashed.

Round of Network Evaluation. A *round of network evaluation* is the process of considering all the nodes of a query network in a certain order, and evaluating each node once, in that order. The significance of the order is that edges added previous to, say, the i-th single node evaluation, are already in F when the i-th evaluation is performed. If evaluations are exhaustive, then the round is an *exhaustive round* of network evaluation.

A Fully Evaluated Network. If a network is such that a round of evaluation applied to it will not add any edge to F, then we say that the network is *fully evaluated.*

Example. The network in Figure 5 is a fully evaluated network. ■

Note that the network is a DAG-network (i.e., a network whose graph is a directed acyclic graph). We revisit this example when we discuss a property of DAG-networks in Section 3.1.

The Problem. The problem for which we propose algorithms is the following. Given a query network (N, F^0) in the input, construct the resulting fully evaluated query network (N, F).

3. NETWORK EVALUATION ALGORITHMS

We propose three algorithms for evaluating query networks. The *Basic* evaluation algorithm is a simple evaluation algorithm which performs rounds of network evaluation until the network is fully evaluated. The *Backward-Radius Triggering* (BRT) evaluation algorithm significantly reduces the number of node evaluations. It does not perform network evaluation rounds. Instead, it identifies nodes whose evaluation is necessary for constructing the fully evaluated network. That is, BRT typically avoids many node evaluations that do not contribute new edges to the network. The *Divide and Conquer* (DAC) evaluation algorithm partitions the network, evaluates each partition separately and merges the results. The partitions' evaluation is completely independent, which makes DAC an algorithm which can greatly benefit from parallelism.

3.1 The Basic Algorithm and Related Results

The first algorithm we present is the *Basic* evaluation Algorithm. This algorithm is a simple algorithm that we use as a baseline for comparison. Pseudo-code for this algorithm is presented as Algorithm 1. Next, we discuss the algorithm and prove two propositions related to the evaluation of query networks in general.

In each iteration, the algorithm performs a single evaluation for each of the nodes in the network. If a round is completed without adding any edge to F, the algorithm stops.

```
Input: $(N, F^0)$, a query network.
Output: $(N, F)$, a fully evaluated query network.
Method:
1 : $stopFlag \leftarrow$ false;
2 : while $stopFlag =$ false
3 :     $stopFlag \leftarrow$ true;
4 :     for each $n$ in $N$
5 :         evaluate $q(n)$ once, let $Q$ be the (binary) result;
6 :         if $Q \setminus F \neq \varnothing$
7 :             $stopFlag \leftarrow$ false;
8 :             $F \leftarrow F \cup Q$;
9 :         end if
10:     end for each
11: end while
```

Algorithm 1: Basic Network Evaluation Algorithm

Example. Let us return to the query network presented in Figure 1. Suppose that this network is evaluated by the Basic evaluation algorithm. In the first round of evaluation, evaluating the query associated with Lisa yields a result relation with one tuple, *(Lisa, Marge)*. This tuple is added to F, and the *stopFlag* is toggled to indicate that the main while loop of the algorithm (lines 2 through 11) has to continue for at least another iteration. In this round, the evaluation of the rest of the nodes results in an empty result set. In the second round (respectively, third round), the evaluation of Bart's node (respectively, Lisa's node) resulted in the addition of the edge *(Bart, Maggie)* (respectively, *(Lisa, Maggie)*), while the evaluation of the rest of the nodes yields no new tuples. In the fourth round, none of the node evaluations results in a non-empty result, and the algorithm stops. ■

The Basic Evaluation Algorithm with Exhaustive Rounds. Note that instead of evaluating each node once, as stated in line 6 of algorithm 1, we can instead exhaustively evaluate each node. This may result in reducing the number of rounds necessary for fully evaluate a network, as discussed in Section 3.1.2.

3.1.1 The Preservation of Cycles Property

PROPOSITION 1. *Consider a query network (N, F^0). Let (N, F) be the fully evaluated network. Then, if (N, F) contains a cycle, (N, F^0) also contains a cycle.*

PROOF. **(Sketch)** Let c be a set of edges which form a cycle in (N, F). If all the edges in c are edges in (N, F^0), then c is a cycle in (N, F^0) and the proposition is proved. If not, then there is at least one edge in c which is in (N, F) but not in (N, F^0). Let us call such an edge a *derived edges.* Among the derived edges in c, consider the <u>last</u> edge that the Basic evaluation algorithm applied to N would add, say (u, v). (u, v) was added as a result of evaluating $q(u)$. Therefore, there is a (directed) path p, from u to v, that does not contain (u, v). We delete (u, v) from c and add the edges of p. c remains a cycle.

We repeat this process until no edges can be removed from c. Every repetition deletes a derived edge from c and replaces it either with non-derived edges or with derived edges that were added to F before the deleted edge. A simple induction shows that eventually, only non-derived edges will remain in the cycle c. □

3.1.2 One-Round Evaluation of DAG Networks

Consider the network used in the example in Section 2, i.e., the network in Figure 4, and assume that it is evaluated using the Basic evaluation algorithm with exhaustive rounds. Note that the network graph is a DAG. If the nodes are evaluated in the order of their ids, then in the first round, the edges *(2,6)* and *(4,8)* are added. In the second round, *(2,8)* is added and the third and last round does not add any edge. However, if the order of evaluation is the reverse of the order of ids, then in the first round, *(4,8)* is added first, and the exhaustive evaluation of node 2 yields *(2,6)* and *(2,8)* (in this order). The second and last round does not add any edge.

Next, we show that a network whose graph is a DAG can be fully evaluated in a *single* exhaustive round of evaluation. Note that knowing that the network is fully evaluated makes a second round redundant.

PROPOSITION 2. *Let (N, F^0) be a query network whose graph is a DAG. (N, F^0) can be fully evaluated in one round of exhaustive evaluation.*

PROOF. **(Sketch)** First, we evaluate leaves (i.e., nodes with no outgoing edges). Then, as long as there are nodes that have not been evaluated, we pick one whose descendants are all evaluated, and exhaustively evaluate it. The DAG structure of the graph ensures that if there is an unevaluated node in the network, then there is also an unevaluated node whose descendants are all evaluated.

An induction shows that after all the nodes are exhaustively evaluated once, the network is fully evaluated. □

3.2 The Backward-Radius Triggering Algorithm

Consider, again, the evaluation of the network illustrated in Figure 1 using the Basic algorithm. Four rounds of evaluation resulted in 28 single node evaluations, whereas only three of these evaluations actually yield addition of an edge. The *Backward-Radius Triggering* (BRT) evaluation algorithm will reach a fully evaluated network by usually performing a significantly lower number of single node evaluations.

BRT takes k, the maximal radius in the network, as input. k is used as a bound on the backward radius of the nodes. In BRT, when an edge, say (u, v), is added, only nodes whose queries can 'sense' the addition are considered for another evaluation. These nodes are such that there exists a (directed) path, whose length is less than the backward radius, between them and u.

Note that the important model feature is that k is small relative to the network. Merely knowing what k is could be ascertained in the first round of BRT. Therefore, passing k as a parameter is not essential to the algorithm. For simplicity, we give k as input to BRT.

Pseudo-code for the BRT evaluation algorithm appears as Algorithm 2. First, all the nodes are put in the set R, and a single node evaluation is performed for every node n in R. This is in fact a round of network evaluation. For every node n whose evaluation results in the addition of an edge (or multiple edges) to F, the set $\{m \in B(n, l) | l < k\}$ is computed (see definition in Section 2) and added to P. P replaces R and the evaluation continues until R is empty.

Input: (N, F^0), a query network; k, maximal query radius.
Output: (N, F), a fully evaluated query network.
Method:

```
1 : $R \leftarrow N$
2 : while $R \neq \varnothing$
2 :    $P \leftarrow \varnothing$
3 :      for each $n \in R$
3 :         evaluate $q(n)$, let $Q$ be the (binary) result;
6 :         if $Q \setminus F \neq \varnothing$
7 :            $P \leftarrow P \cup \{m \in B(n, l) | l < k\}$
8 :            $F \leftarrow F \cup Q$;
9 :         end if
10:      end for each
11:    $R \leftarrow P$
12: end while
```

Algorithm 2: Backward Radius Triggering Algorithm

Example. Consider again the query network presented in Figure 1. In this network, $k = 3$. If evaluated with the BRT algorithm, the first iteration will consider all the nodes for a single evaluation, and the edge *(Lisa, Marge)* will be added. $B(Lisa, 1) = \{Bart\}$ and $B(Lisa, 2) = \varnothing$. Therefore, $P = \{Bart\}$. The evaluation of the single node in P results in the addition of the edge *(Bart, Maggie)*, and at the end of this iteration, $P = \{Lisa\}$. In the next iteration, the edge *(Bart, Maggie)* is added, and $P = \{Bart\}$. In the next iteration, no edge is added. As a result, $P = R = \varnothing$ and the algorithm stops.

Comparing Basic and Backward-Radius Triggering
The total number of single-node evaluations in the latter example is (broken by iteration) $7 + 1 + 1 + 1 = 10$. As shown above, the Basic algorithm performs, on the same network, 28 evaluations. The benefit of saving single rule evaluations come at the price of computing the sets $B(n, k)$. Also, Basic does not use k.

3.3 The Divide and Conquer Algorithm (DAC)

We would like to be able to process large networks. In the DAC evaluation algorithm, we take advantage of the clustered nature of social networks in order to partition the network into networks of more manageable size. Generally speaking, social networks have a structure in which participants have more links to participants within their community than to individuals from other communities [18] (see more in Section 6). We use existing knowledge and algorithms for graph partitioning in order to partition the graphs to parts with a relatively small number of edges between them. This partitioning enables us to fully process small, dense sub-networks, taking advantage of locality of reference and minimizing work for a merge step.

A partitioning algorithm for a query network takes a query network, say (N, F^0), as input and produces a number of query networks as output. N is partitioned into (non-overlapping) sets of nodes. Each such set N_i, and the edges in F^0 between the nodes in N_i form a query network in the output. *Crossing edges* are edges in F^0 that are in none of the created networks.

DAC takes a graph-partitioning algorithm and the number of parts to partition to as input. In addition, like BRT, DAC takes k, the maximal radius of query in the network, as input.

Pseudo code for DAC appears as Algorithm 3. DAC operates as follows. First, the partition algorithm partitions the network into smaller networks. Every part is evaluated separately using the BRT evaluation algorithm (Line 2). Then, a match-making procedure is invoked (Line 4). Two networks such that the number of crosiing edges between them is maximal are matched. Then, the rest of the networks are considered, and another pair is matched and so on (see procedure *mergePairs*). The match making continues until less than two networks remain unmatched.

Each pair is merged into one network (Line 7). The nodes of the new network are the union of the nodes of the networks being merged. The edges are the union of the edges of the networks being merged, as well as the crossing edges between the merged networks. Due to the addition of crossing edges, the merged network is not fully evaluated. The *merge&eval* procedure evaluates the merged network, first by evaluating all the nodes n that are sources of the cross edges and the nodes within $B(n, l), l < k$ for each such node n, in order to include all the nodes in their backward radius. Like in BRT, any such node whose evaluation results in the addition of new edges triggers the evaluation of the nodes potentially in their backward radius and so on until a fixpoint is reached.

```
Input: $(F^0, N)$, a query network; $k$, maximal query radius;
$A$, a graph-partitioning algorithm; $p$, number of partitions.
Output: $(F, N)$, a fully evaluated query network.
Method:
1 : invoke $A$ to partition $N$ into $p$ parts; let $R$ be the set of
- : parts, $\{(F_i^0, N_i) \mid 1 \leq i \leq p\}$;
2 : invoke BRT evaluation on each $r \in R$;
3 : while $\exists (u, v) \in F^0$ s.t. $u$ and $v$ are in different parts
4 :     $P \leftarrow matchPairs(R)$; //see procedure below
5 :     for each pair $(r_i, r_j) \in P$
6 :         $R \leftarrow R \setminus \{r_i, r_j\}$;
7 :         $R \leftarrow R \cup merge\&eval(r_i, r_j)$;//see procedure below
8 :     end for each
9 : end while
```

Algorithm 3: Divide and Conquer Algorithm.

```
procedure matchPairs($R$) //$R$ is a set of networks
1 : for each crossing edge $(u, v)$ w.r.t. the networks in $R$
2 :     let $r_u$ (respectively, $r_v$) be $u$'s (respectively, $v$'s)
- :     network;
3 :     initialize a counter $c_{u,v}$ to 0, if not exists;
4 :     $c_{u,v} \leftarrow c_{u,v} + 1$;
5 : end for each
6 : $P \leftarrow \varnothing$;
7 : while there is more than one network in $R$
8 :     add to $P$ a pair $(r_v, r_u)$ s.t. $c_{u,v} + c_{v,u}$ is maximal;
9 :     delete $r_v$ and $r_u$ from $R$;
10: end while
11: return $P$;
end procedure
```

Example. Consider the network sketches in Figure 6. Dark grey (respectively, light grey) represents a fully evaluated (respectively, non fully evaluated) network.
In *(a)*, a network partitioned into four parts is presented. Only crossing edges are presented. In *(b)*, each of the four parts is fully evaluated, ignoring crossing edges. In *(c)*, the matchmaking result is illustrated. The parts with maximal number of crossing edges between them were paired. Note that the pairs of networks are not yet evaluated. In *(d)*, the merged and fully evaluated pairs are shown. Another matchmaking step is *(e)*, and the fully evaluated network is *(f)*.

Note that as in BRT, k need not be known in advance. Here also, k can be ascertained for each initial partition (i.e., before any merge has been done) on the first round and maintained for further evaluations as the maximum value of each merged pair (note however that k for different parts may be lower than the global k). For simplicity, we give k as input to DAC.

4. EXTENSIONS

The model presented so far is the minimal model needed to prove our concept of large queries in a Web 2.0 environment. However, in order to be used in real systems, some extensions to the model should be considered. Larger-scope additions, which require thorough definitions are mentioned in Section 7.

```
procedure merge&eval($(N_1, F_1)$,$(N_2, F_2)$)
1 : $N \leftarrow N_1 \cup N_2$;
2 : $F \leftarrow F_1 \cup F_2$;
3 : let C be the set of crossing edges between
- : the two networks;
4 : $F \leftarrow F \cup C$;
5 : $M \leftarrow \{n_1 | (n_1, n_2) \in C\}$;//sources of new edges
6 : $G \leftarrow \{g | g \in B(m, l), 1 < l < k, m \in M\}$;//nodes
- : possibly affected by the new edges
7 : for each $m \in M$
8 :     evaluate $q(m)$, let $Q$ be the (binary) result;
9 :     $F \leftarrow F \cup Q$;
10: end for each
11: while $G \neq \varnothing$
12:     $P \leftarrow \varnothing$;
13:     for each $g \in G$
14:         evaluate $q(g)$, let $Q$ be the (binary) result;
15:         if $Q \setminus F \neq \varnothing$
16:             $P \leftarrow P \cup \{m \in B(g, l) | l < k\}$
17:             $F \leftarrow F \cup Q$;
18:         end if
19:     end for each
20:     $G \leftarrow P$;
20: end while
21: return $(N, F)$;
end procedure
```

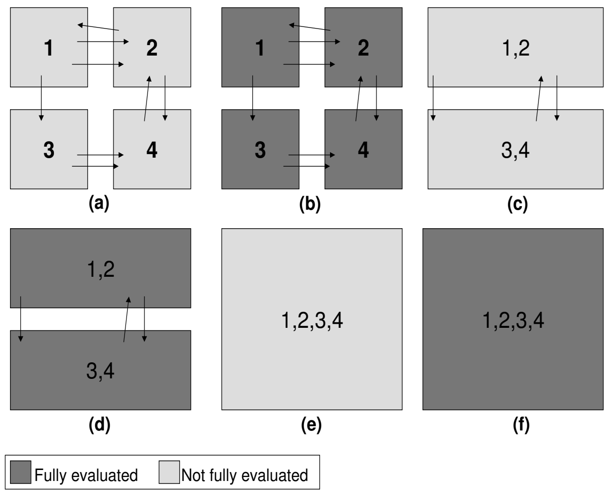

Figure 6: Divide and Conquer Example.

Data integration. The web environment is diverse in data types. Of particular interest is the interplay between structured and unstructured data [3]. A natural extension to our model will be to also process XML data. Every node in the model, in addition to having a list of friends, will also have an XML document associated with it. Inspired by [27] and [26], a query that refers to the XML document and to the structured data will have the following form:

```
F⁺(n,X)←F(n,Y),F(Y,X),F(n,Z),F(Z,X),
          xpath(X,'/data/lanaguage[text()="French"]')
```

The *xpath* predicate will be satisfied by nodes whose associated XML document satisfies the expression *'/data/lanaguage[text()="French"]'.*
As in SQL/XML [28], the expression is evaluated at the document root node. Query languages other than XPath can be used to query XML data, and other data types with their query languages may be used in the CQ model.

More than one query. A realistic policy for the additions of connections to friends' lists in a social network will most likely require more than one query. A natural extension of our model would be to have a set of queries, or even a datalog program (or programs), associated with each node. This addition raises interesting evaluation problems. For example, how to order the evaluation of queries associated with the same node so as to minimize the number of evaluation rounds. Also, the radius and backward radius can not be defined as they are currently defined.

Not using a global radius. Currently, the radius is a parameter of an evaluated network or part of it. As explained above, the radius is either gathered from the network or passed as input. However, we use the radius as a bound on the backward radius of network nodes. A more sophisticated algorithm will efficiently initialize and maintain for every node its backward radius, and avoid evaluating nodes that are within the (globally) highest backward radius in the network when it is unnecessary for the particular node given its backward radius.

5. EXPERIMENTS

5.1 Implementation

We implemented the presented algorithms in a system. The system was programmed in Java, using the open-source DBMS MySQL, version 5.0. The experiments were carried out on a machine with a Pentium CPU of 1.5GHZ and 1GB of RAM, running the Windows XP operating system. We experimented with the algorithms, both on synthetic query networks as well as on query networks derived from the DBLP data, which reflect social behavior. The system code, dataset generation code and the resulting datasets are available for repeatability testing.

5.2 Synthetic Datasets

Our synthetic datasets are built as follows. Consider the illustration in Figure 7. The figure shows a number of clusters. Connections may exist between participants in the same cluster or between participants in neighboring clusters. The probability that participants in the same cluster are connected is higher than the probability that two participants in neighboring clusters are connected. The clusters form a cycle in which every city has exactly two neighbors, one to the right and one to the left.

The graphs are synthesized as follows. The number of clusters and the number of nodes in each cluster are given as input. For every node n_1, we pick at random a node n_2 s.t. $n_1 \neq n_2$ from the same cluster, and add an edge (n_1, n_2). Then, every node has probability α to be connected to each

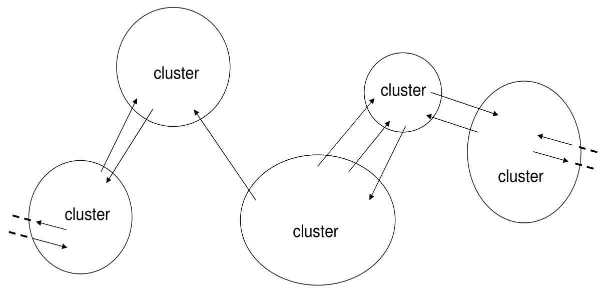

Figure 7: Illustration of the Synthetic Datasets Structure.

of the other nodes in the cluster. At least one node, plus up to additional β nodes per hundred in a cluster, are connected to a randomly chosen participant in the immediate neighboring cluster to the left, and a similar number is connected to participants in the immediate neighboring cluster to the right.

The queries used in our experiments are q_a and q_b from the example in Section 2, as well as an unsatisfiable query. Partitioning for the DAC algorithm is done using the Metis software package [21]. We also show results for partitioning the graph to parts such that each part corresponds to a cluster.

Experiment 1. In this experiment, we create 10 datasets. The i-th dataset has $5 * i$ clusters. Each cluster has 160 nodes. $\alpha = 1/200$, $\beta = 2$. Nodes are randomly associated with either q_a or q_b (with probability 0.5 for each of the queries).

Consider Figures 8 and 9. The running time results for each of the ten datasets appear in Figure 9 as four figures (from left to right): DAC time using the Metis partitioner (DAC-Metis), DAC time using the initial, predefined partition to clusters (DAC-Pre), BRT time, Basic time. The datasets are ordered according the IDB/EDB ratio, which is the number of added edges divided by the number of original EDB edges. The datasets numbers correspond to the table in Figure 8.

Basic evaluation is the slowest, except for the smallest dataset. As for the remaining two algorithms, the results clearly show that DAC outperforms BRT when the number of added edges is high relative to the number of EDB edges. BRT is better when relatively few edges are added in reaching a fully evaluated network. Partitioning using Metis or partitioning according to the initial clusters does not make a significant difference for DAC's performance (partitioning times are included in the total DAC time). Figure 10 orders the same results by the number of added edges (namely, the IDB size). Figure 11 orders the results by the size of the EDB. The results show a clear correlation between the IDB/EDB size ratio and the benefit of using DAC rather than BRT. A similar correlation exists between the IDB size

Set	Nodes	EDB	IDB	IDB/EDB
1	800	1306	32	0.04
2	1600	2887	3699	2.311875
3	2400	4335	3623	1.509583
4	3200	5822	826	0.258125
5	4000	7291	4255	1.06375
6	4800	8749	386	0.080417
7	5600	10200	489	0.087321
8	6400	11650	476	0.074375
9	7200	13078	543	0.075417
10	8000	14588	673	0.084125

Figure 8: Experiment 1 datasets.

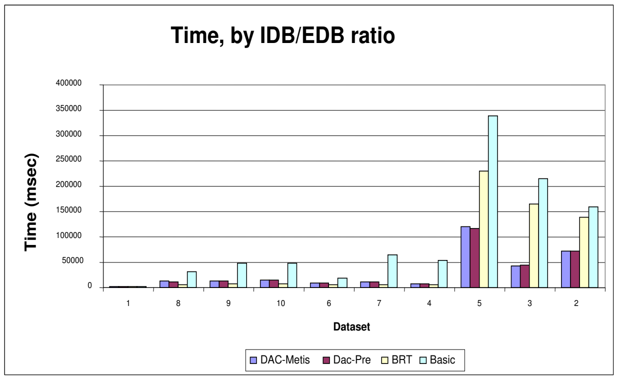

Figure 9: Time results of Experiment 1. Datasets ordered by IDB/EDB ratio.

and the benefit of using DAC.

Experiment 2. In this experiment, we create 10 datasets. Again, the i-th dataset has $5 * i$ clusters. Each cluster has 150 nodes. $\alpha = 5/900$, $\beta = 2$ and q_a and q_b are independently and evenly distributed among the nodes. Figure 12 shows the time results for the three algorithms (partitioning

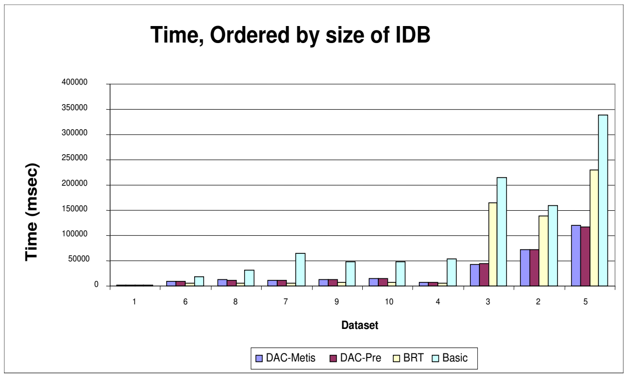

Figure 10: Time results of Experiment 1. Datasets ordered by IDB size.

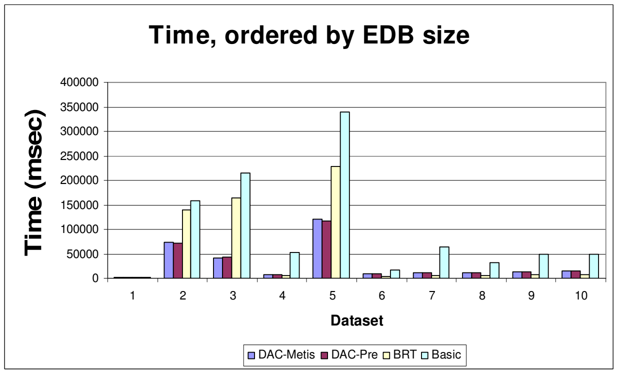

Figure 11: Results of Experiment 1. Datasets ordered by EDB size.

1	0.06876
2	0.07945
3	0.086699
4	0.130908
5	0.198418
6	0.322482
7	0.550814
8	0.540871
9	0.682822
10	0.988547

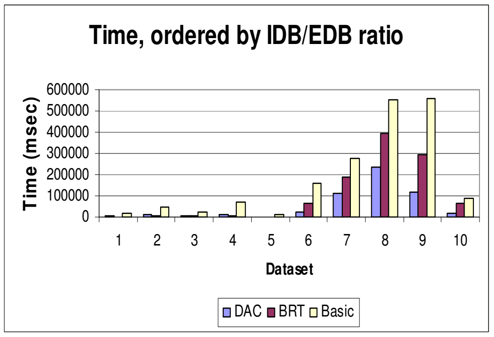

Figure 12: Results of Experiment 2. The table shows IDB/EDB ratio per dataset. Datasets are ordered by IDB/EDB ratio.

for DAC is done using Metis). The results are ordered by the IDB/EDB ratio, which also appears in a table for each dataset. Here too, we see that Basic is the slowest algorithm, and that the benefit of using DAC over BRT is correlated with the IDB/EDB ratio.

Experiment 3. In this experiment, each cluster has 225 nodes, $\alpha = 1/200$, $\beta = 2$ and q_a, q_b and an unsatisfiable query are evenly distributed among the nodes. Figure 13 shows the time results of the three algorithms (partitioning for DAC is done using Metis). The results are ordered by the values of the IDB/EDB ratio, which also appear in the table. Here too, we see that the benefit of using DAC over BRT is correlated with the IDB/EDB ratio.

Discussion. The choice of the best algorithm for evaluating a given network is greatly affected by the size of the IDB. We can see that sometimes large EDBs can be evaluated faster using BRT as long as their IDB is small. On the other hand, DAC clearly performs better for networks with large IDBs. This result suggests the need for a cost model for choosing an algorithm for evaluating query networks.

5.3 DBLP Datasets

We also conducted experiments on data derived from the DBLP datasets [12]. We use DBLP in order to capture pat-

1	0.106947
2	0.482495
3	0.050733
4	0.624307
5	0.658115
6	0.71474
7	0.927652
8	1.336794
9	2.620269

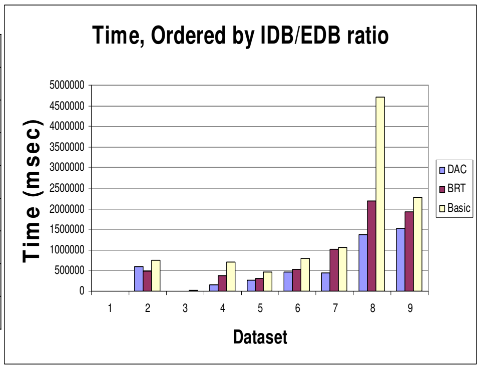

Figure 13: Results of Experiment 3. The table shows IDB/EDB ratio per dataset. Datasets are ordered by IDB/EDB ratio.

1	0.081467
2	0.330961
3	1.348783
4	1.380309
5	1.587966
6	1.597908

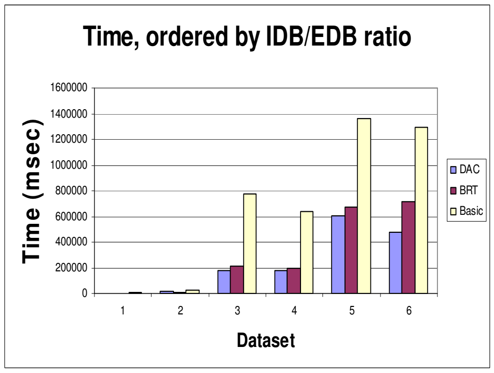

Figure 14: Results of Experiment 4. The table shows IDB/EDB ratio per dataset. Datasets are ordered by IDB/EDB ratio.

terns of social behavior that are reflected in the collaborations between authors. We extracted 260360 papers with their 260801 authors. For each pair of authors we also keep the number of papers that they published together.

We construct the datasets for the experiments as follows. We start with one author, and perform a crawling-like procedure. We retrieve five of the author's collaborators (if there are less than five, all are retrieved). Then, we retrieve five collaborators of each of the five retrieved in the first step, and so on until the desired number of authors is reached.

The input network is constructed as follows. Each author is represented by a node. If the number of publications in common to u and v is at least α of v's total number of publications then there is an edge from u to v. That is, if u participated in a high enough share of v's publications (α percent or more), the edge (u, v) is added.

Experiment 4. In this experiment, the graph consists of 1300 nodes, partitioned into 16 parts (using Metis). We create the datasets by gradually lowering α from 33% to 14%, in order to increase the number of edges in the EDB. Figure 14 shows the running time results of the three algorithms on datasets generated as described above. The IDB/EDB ratio also appears in the table in Figure 14. Here too, we see that the advantage of evaluating with DAC is higher as the IDB/EDB ratio is higher.

Experiment 5. In this experiment, the graph consists of 1400 nodes, partitioned into 16 parts (using Metis). Again, we create the datasets by gradually lowering α from 33% to 14%. Figure 15 shows the result of this experiment, which are in line with the results of Experiments 1 through 4. Results for the Basic algorithm were not included in the figure and are instead given in the appendix, so as not to overload the figure with high numeric values.

1	0.062775
2	0.353121
3	0.398953
4	0.439271
5	0.707170
6	2.392594

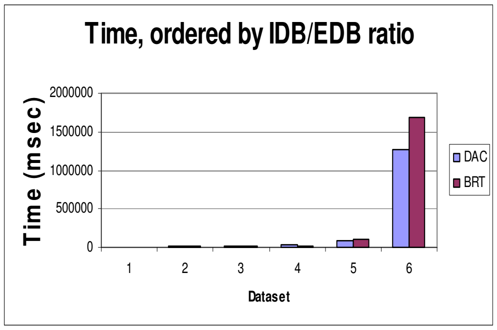

Figure 15: Results of Experiment 5. The table shows IDB/EDB ratio per dataset. Datasets are ordered by IDB/EDB ratio.

6. RELATED WORK

The structure and growth patterns of social networks are topics that have been studied in many contexts, from physics and biology related systems to transportation, telephony and internet networks [18]. [18] also reviews several models for network growth. Many recently published works in the area of social networks are concerned with analysis of social networks structure, privacy and security in social networks, search related issues and many more. In SIGMOD record, March 2008, as part of the report on the Databases and Web 2.0 Panel at VLDB 2007 [4], it is stated that "Understanding and analyzing trust, authority, authenticity, and other quality measures in social networks will pose major research challenges." In fact, adding queries, or programs, to nodes can be an extension for these topics to the actual running of a technology-assisted social network.

Historically, the evolution of information and structure in social networks has been a subject of research for more than a hundred years. See [31] for an overview of the subject. In particular, the interactive applet at [22] provides an interesting view as to inner workings of such networks. Of course, the Internet has brought social networks to the forefront. Perhaps, the most distinct feature of these networks is that they are centrally managed by and relatively open to new participants. It is not clear whether parameters such as Dunbar's number [13, 14], a limit on the number of individuals with whom a person can consistently interact over time (believed to be around 150), is still valid in the WWW context. Conceivably, technology can assist in marinating more meaningful connections or even many weak but useful connections.

Related work in the area of Datalog evaluation and optimization is thoroughly covered in [29] and [1]. However, these assume small queries and large data. Processing of Web and Internet data with datalog-like formalisms is studied in [8], [2], [26], [19]. We are not aware of work regarding very large queries or regarding Datalog with a massive number of rules.

7. CONCLUSIONS AND FUTURE WORK

We introduce the *Query Network* model, inspired by the development of Web 2.0 applications. The queries in the network are created by a large number of participants. Together, they form a datalog program whose size is in the order of magnitude of the size of the data. We implemented algorithms to evaluate query networks and experimented with them on synthetic datasets and on datasets derived from DBLP. The experiments demonstrate the effectiveness of our methods.

As future work, we plan to study and implement the extensions discussed in Section 5. As additional future work, we plan to study the following topics: *(1)* The deletion of nodes from the network; *(2)* The parallelization of the suggested algorithms, and in particular the parallelization of the DAC algorithm; *(3)* Introducing a more complex datalog-based model which captures the process of reaching an agreement between network participant before adding a network connection and *(4)* Introducing different types and weights to the possible connections in the network.

8. REFERENCES

[1] S. Abiteboul, R. Hull, V. Vianu. Foundations of Databases. Addison-Wesley 1995.

[2] S. Abiteboul, Z. Abrams, S. Haar, T. Milo: Diagnosis of asynchronous discrete event systems: datalog to the rescue!. In PODS 2005.

[3] R. Agrawal et. al. The Claremont Report on Database Research. Berkeley, California, 2008.

[4] S. Amer-Yahia, V. Markl, A. Y. Halevy, A. Doan, G. Alonso, D. Kossmann, G. Weikum: Databases and Web 2.0 panel at VLDB 2007. SIGMOD Record 37, 2008.

[5] F. Bancilhon, D. Maier,Y. Sagiv, J.D. Ullman. Magic sets and other strange ways to execute logic programs. In PODS 1986.

[6] The Britannica Online Encyclopedia. http://www.britannica.com.

[7] F. Arni, K. Ong, S. Tsur, H. Wang, C. Zaniolo: The Deductive Database System LDL++. TPLP vol. 3(1), 2003.

[8] R. Baumgartner, S. Flesca, and G. Gottlob. The E-log Web Extraction Language. In LPAR 2001.

[9] F. Bry, T. Furche, B. Linse, A. Schroeder. Efficient Evaluation of n-ary Conjunctive Queries over Trees and Graphs. In WIDM 2006 (in conj. with CIKM06).

[10] A. Chandra, P. Merlin, Optimal implementation of conjunctive queries in relational databases. 9th ACM Symposium on the Theory of Computing, 1997.

[11] S. Cohen, Y. Kanza and Y. Sagiv. SQL4X: A Flexible Query Language for XML and Relational Databases. In DBPL 2001.

[12] The DBLP Computer Science Bibliography, 2008. http://www.informatik.uni-trier.de/ ley/db/

[13] R.I.M. Dunbar: Neocortex size as a constraint on group size in primates. In Journal of Human Evolution 22, 1992.

[14] R.I.M. Dunbar: Coevolution of neocortical size, group size and language in humans. In Behavioral and Brain Sciences 16(4), 1993.

[15] Facebook Social Utility. http://www.facebook.com, 2008.

[16] Facebook Query Language. http://developers.facebook.com, 2008.

[17] S. Jain, R. Mahajan, D. Suciu: Translating XSLT programs to Efficient SQL queries. In WWW 2002.

[18] E. M. Jin, M. Girvan, M. E. J. Newman: Structure of growing social networks. Physical Review E, Vol. 64, 2001.

[19] W. May. XPath-Logic and XPathLog: A Logic-Programming Style XML Data Manipulation Language. TPLP 4(3): 239-287, 2004.

[20] LinkedIn, Professional Network Management. http://www.linkedin.com, 2008.

[21] Metis - Family of Multilevel Partitioning Algorithms. Karypis Lab, University of Minnesota. http://glaros.dtc.umn.edu/gkhome/views/metis, 2008.

[22] Modeling self-organization of communication and topology in social networks. http://cmol.nbi.dk/models/inforew/inforew.html

[23] MySpace (a place for firends). http://www.myspace.com, 2008.

[24] S. Naqvi and S. Tsur: A Logical Language for Data and Knowledge Bases. C.S. Press, New York 1989.

[25] R. Ronen and O. Shmueli. Conjunctive Queries over DAGs. In NGITS 2006.

[26] R. Ronen and O. Shmueli. Evaluation of Datalog Extended with an XPath Predicate. In WIDM 2007 (in conj. with CIKM07).

[27] R.Ronen and Oded Shmueli. Using a Relational Processor and an XPath Processor to Evaluate Joint Queries. EDBT Workshops, 2008.

[28] International Organization for Standardization (ISO). I.T. DB Language SQL Part 14: XML-Related Specifications (SQL/XML).

[29] J.D. Ullman. Principles of Database and Knowledge Base Systems, Vol. I,II. C.S. Press, Rockville, Md, 1989.

[30] Wikipedia - The Free Encyclopedia. http://en.wikipedia.org/.

[31] Wikipedia entry on Social Networks. http://en.wikipedia.org/wiki/Social_network.

[32] XPath 1999. http://www.w3.org/TR/xpath.

[33] XQuery 1.0. http://www.w3.org/TR/xquery.

9. APPENDIX

Results for the Basic algorithm in Experiment 5 are: Dataset 1: 6489 msec, Dataset 2: 30414 msec, Dataset 3: 36833 msec, Dataset 4: 56041 msec, Dataset 5: 190054 msec, Dataset 6: 2771465 msec.