Scalability of source identification in data integration systems

François Boisson, Michel Scholl, Imen Sebei, and Dan Vodislav

CNAM/CEDRIC, Paris, France
francois.boisson@gmail.com, {scholl,imen.sebei,vodislav}@cnam.fr

Abstract. Given a large number of data sources, each of them being indexed by attributes from a predefined set $\mathcal{A}$ and given a query q over a subset Q of $\mathcal{A}$ with size k attributes, we are interested in identifying the set of all possible combinations of sources such that the union of their attributes covers Q. Each combination c may lead to a rewriting of q as a join over the sources in c. Furthermore, to limit redundancy and combinatorial explosion, we want the combination of sources to produce a *minimal* cover of Q. Although motivated by query rewriting in OpenXView [3], an XML data integration system with a large number of XML sources, we believe that the solutions provided in this paper apply to other scalable data integration schemes. In this paper we focus on the cases where the number of sources is very large, while the size of queries is small. We propose a novel algorithm for the computation of the set of minimal covers of a query and experimentally evaluate its performance.

1 Introduction

The tremendous multiplication of data sources at every level (personal, enterprise, communities, web) in the past few years raises new challenges in querying such heterogeneous and highly distributed data.

In this context, we address a general problem when querying a large number N of data sources. Assume each source is indexed by attributes from a predefined set $\mathcal{A}$. Given a query q over a subset Q of $\mathcal{A}$ with k attributes, we want to find the sources that can partially or totally answer the query. If the source partially answers the query then it might, by a join with other sources, totally answer the query. Thus we are interested in identifying the set of all possible combinations of sources such that the union of their attributes covers Q.

The meaning of attributes is very general here. A data source indexed on an attribute provides data related to that attribute. In concrete cases attributes may be elements of a mediation schema connected to the data source through various mappings, or simple keywords that characterize the data source content.

Each combination of sources that covers Q may lead to a rewriting of q as a *join* over these sources. We consider here that joins between sources are realized in *a predefined way* and are not explicitly given in query q or in mappings. E.g., implicit joins between data sources may be fusion joins on key attributes [1, 3], natural joins for relational sources [5], joins based on links, etc.

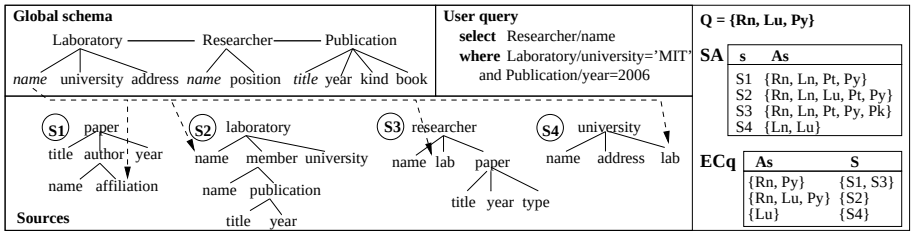

Fig. 1. An example of data integration system

Furthermore, the number of possible combinations of sources being potentially very large, we made the choice to limit combinatorial explosion and redundancy by only considering combinations that produce *a minimal cover* of Q. In a minimal cover, no source can be removed without breaking the cover of Q, i.e. each source covers some attribute in Q that no other source in the combination covers. Intuitively, the more attributes of a query are found in a source, the more pertinent the source is for that query. Minimal covers limit dispersion of query attributes among sources, which would produce less pertinent results.

This work on source identification was motivated by data integration systems dealing with a large number of (relational or XML) local-as-view [8] sources, where queries written on a common schema have to be translated into queries on the sources. Searching by keywords on the web is another class of applications that we have in mind. The idea is to extend current search engines, that limit answers to single documents containing (part of) the keywords so as to consider joins based on hyperlinks between documents.

More specifically, this work was initiated when studying the query rewriting algorithm of OpenXView [3], a data integration model for a large number of XML sources. In OpenXView a query expressed on an ontology is translated into a union of joins of pattern tree queries on the sources (each source may have a different DTD or XML Schema), thanks to a mapping between ontology concepts and XML elements.

Although the mechanism studied in this paper applies to a large variety of mediation schemes, we chose to illustrate it on an example of OpenXView data integration, showed in Figure 1. Sources S_1-S_4 contain XML documents about research laboratories, researchers and their publications, whose schemas are presented in a tree-like form. The global schema is an ontology with concepts *Laboratory*, *Researcher* and *Publication*, each one with several properties that form a tree, those in italic being *key* properties used for implicit joins (e.g. *name* for *Researcher*). In this case, the set of attributes $\mathcal{A}$ is the set of all properties (paths) in the global schema: $\mathcal{A}$={*Laboratory/name, Laboratory/address,* ...}. The indexing of sources with attributes is realized through node-to-node mappings between attributes and source schema elements, e.g. *Laboratory/name* is mapped to *paper/author/affiliation* in S_1, to *laboratory/name* in S_2, etc. For the sake of readability, only a few mappings are shown.

An OpenXView query q is a set of projections and selection conditions over elements of $\mathcal{A}$. The query in Figure 1 asks for names of researchers from MIT that published in 2006; the set of query attributes is $Q = \{$*Researcher/name, Laboratory/university, Publication/year*$\}$. The source identification problem is to find all combinations of sources in S_1-S_4 that produce a minimal cover of Q. E.g., combination (S_1, S_4) produces a minimal cover of Q, because S_1 covers the researcher name and the publication year, while S_4 covers the remaining attribute in Q (university name), but (S_1, S_2) is not minimal, because S_2 covers all attributes in Q and S_1 becomes redundant. Note that source identification is just the first step in query rewriting prior to actual query translation. This paper focuses on this critical step. We address cases where the number of sources may be very large (1000 or more), while the query size is small (no more than 8 attributes). In a previous paper [3], we designed a simple algorithm, referred to in the following as SI_1, for identifying the relevant sources, which scans all the sources and progressively constructs the minimal combinations. In the following, we propose an efficient and scalable algorithm for source identification, called SI_2, based on the pre-computation of minimal covers and appropriate when the number of query attributes k is ≤ 8, which in most applications is a reasonable assumption. We experimentally compare the time performance of this algorithm and that of algorithm SI_1 and show that SI_2 drastically outperforms SI_1. However SI_2 requires a significant amount of space for storing the pre-computed sets of minimal covers.

The paper is organized as follows. Section 2 discusses related work, then Section 3 presents both algorithms SI_1 [3] and SI_2. Section 4 reports on our performance study and some conclusions are drawn in Section 5.

2 Related Work

We focus on the integration of a *large* number of sources, where publishing of new sources is frequent and unpredictible. We deal with cases where implicit joins between sources exist, hence cases with few restrictions on combining sources to answer the query. Moreover, we aim at simplifying the expression of queries by the final user: querying with selection and projection of attributes, which does not introduce strong constraints for combining sources. We limit the number of combinations to a subset with reasonable size: *minimal* combinations. Our goal is to find fast algorithms for computing them, scalable with the number of sources N. The next step, ranking over query rewritings obtained from minimal combinations, is out of the scope of this paper. Although source identification is a well known issue in querying data integration systems, to our knowledge, existing algorithms for source selection are not appropriate in our context:

In global-as-view systems [6, 2, 7], joins between sources are defined in mappings, which makes source selection trivial, but maintaining mappings when adding new sources is difficult. Attempts to improve maintainability, such as that proposed in [11], where joins are moved to an intermediate level of the global schema, are not enough to make global-as-view systems appropriate for large scale integration with frequent changes.

Local-as-view architectures are a better match for our needs. But existing systems are based on specific models and come with limitations when adapted to our context. The best known query rewriting algorithms are Bucket [5] and its optimization MiniCon [10], designed for relational integration systems. In these algorithms, not scalable with N, even if implicit joins between sources exist, the query model introduces strong constraints for source combinations. If the general Bucket strategy can be adapted to our context ([3] shows that our SI_1 algorithm outperforms a scalable variant of Bucket), MiniCon is too dependent on its underlying model to be reused for other data integration models.

Among the other local-as-view approaches, [12] proposes an integration framework for XML and relational data which does not fit our context since joins are not implicit, but expressed in the global schema. [4] studies the contribution of sources to a given query in integration systems, i.e. their classification as self-contained or complementary to other sources, but does not compute source combinations. In contrast, [1] uses implicit joins based on keys, but its recursive algorithm does not scale with N, nor consider minimality.

3 The Algorithms

3.1 Introduction and Notations

Let $\mathcal{S}$ denote the set of sources whose size is N. Indexing of data sources with attributes of $\mathcal{A}$ is represented by (N1NF) relation SA. SA is a set of couples $[s, A_s]$, where $s \in \mathcal{S}$ is a source and A_s is the set of attributes in $\mathcal{A}$ provided by s. For the example in Figure 1, the subset of attributes of the query Q and SA are shown on the right side of the figure. We use an abbreviated notation for the attribute names, e.g. *Rn* (*Py*) stands for *Researcher/name* (*Publication/year*).

Both source identification algorithms presented in this section have two steps:

1. The first step, common to both algorithms, takes as inputs the query q and $SA(s, A_s)$ and gives as an output $EC_q(A_s, S)$ the set of couples where the first component $A_s \subseteq Q$ is a non-empty subset of attributes of the query and the second component $S \subseteq \mathcal{S}$ is the subset of sources providing *exactly* the attributes in A_s.
   There are at most $2^k - 1$ tuples in relation EC_q where k is the number of attributes in the query. Basically for a query, the set of sources is split into equivalence classes according to the attributes provided by the source. Tuples of EC_q represent all non empty equivalence classes for query q.
   Because of space limitations we do not detail here the $O(kN)$ algorithm for step 1. The algorithm scans the N sources in SA and for each $[s, A_s] \in SA$, it computes $A_q = A_s \cap Q$ (in $O(k)$ using a hash table for A_s). If $A_q \neq \emptyset$, source s is added to EC_q in the set of sources corresponding to the set of attributes A_q (constant time if $EC_q(A_s, S)$ is a hash table with key A_s).
   For the example in Figure 1, the intersection with Q of attribute sets in SA produces three distinct sets, each one accounting for a tuple in EC_q. E.g., source S_4 covers only attribute Lu in Q, while S_1 and S_3 cover both exactly Rn and Py, thus belong to the same equivalence class. Compared

$\mathbf{SI}_1(EC_q,\, Q)$	$\mathbf{MC}(c,\, E,\, MA,\, Q)$	$\mathbf{minimal}(c,\, i,\, MA)$
output : set of minimal covers	output : set of minimal covers	output : boolean
Begin	**Begin**	**Begin**
$c_0 = (\,)$	If $(MA \stackrel{s}{=} Q)$ return($\{c\}$)	If $Attr(i) \subset MA$
$E_0 = \pi_{A_s}(EC_q)$	$\mathcal{C}$=$\emptyset$	return(false)
$MA_0 = \emptyset$	While $E \neq \emptyset$ repeat	End If
return MC(c_0, E_0, MA_0, Q)	i=pop(E)	$NewMA$=$MA \cup Attr(i)$
End SI_1	If minimal(c, i, MA)	For each $m \in c$ repeat
	$\mathcal{C}$=$\mathcal{C} \cup$ **MC**(c+i, copy(E), $MA \cup Attr(i)$, Q)	If $(NewMA - Attr(m)) \stackrel{s}{=} NewMA$
	End If	return(false)
	End While	End If
	return $\mathcal{C}$	End For
	End MC	return true
		End $minimal$

Fig. 2. SI_1 algorithm with recursive minimal covering (MC) and minimality test

to the number of sources N which might be larger than one thousand, the maximal number of equivalence classes keeps small for reasonable values of the number of attributes in a query k (with $k = 6$ it is at most 63).

2. The second step takes as an input EC_q and computes $\mathcal{C}$ the set of minimal covers of sources (mcs). An mcs is a cover of the attributes in Q, i.e. a set of one or more classes (tuples of EC_q) such that the union of the query attributes in each element (*member*) of the set is equal to Q. Furthermore as aforementioned, we require this cover to be minimal, i.e. if we remove any member of the mcs, then Q is not anymore covered by the attributes in the mcs. Once the set of mcs has been computed then the source identification is over: each mcs represents a possible combination of sources. Let $m_1, ..., m_n$ be the n members of an mcs. Then $s_1, ..., s_n$ where $s_i \in m_i$ for $i \in \{1, ..., n\}$, is a potential combination of sources for query translation.
   Since for a given query with size k attributes, $|\mathcal{C}_k|$, the maximal number of mcs, is exponential in k[1], one expects this second step to be expensive.

We present below two algorithms for step 2 and compare their performance in the following section.

3.2 Algorithm $\mathbf{SI_1}$

This algorithm (Figure 2), introduced in [3], recursively generates all minimal covers of Q with equivalence classes in EC_q. The recursive algorithm MC takes as inputs, a minimal partial cover c (to be completed to a minimal cover), the ordered set of equivalence classes E not yet considered in building c, the multiset MA of query attributes covered by c (counting for each attribute the number of times it is covered), and the target query attributes Q to cover.

The initial call to MC starts with an empty cover and with all the equivalence classes extracted from EC_q. Each call to MC tests whether c is a new solution

[1] The number of minimal set covers of the set of integers K={1..k} is [9]:
$| \mathcal{C}_k | = \sum_{\ell=1}^{\ell=k} \mu(k, \ell)$, where $\mu(k, \ell)$ is the number of minimal covers of K={1,..,k} with ℓ members:
$\mu(k,\ell) = \frac{1}{\ell!} \sum_{m=\ell}^{min(k, 2^\ell - 1)} C_{m-\ell}^{2^\ell - \ell - 1} m! s(k, m)$,
where s(k,m) is a Stirling number of the second kind. E.g.: $| \mathcal{C}_4 | = 49$, $| \mathcal{C}_6 | = 6424$, $| \mathcal{C}_8 | = 3732406$

```
SI2(members, MSCk)
     output: valid, all bits initially set to 1
Begin
    For each As in P({1,...,k}) − {∅} loop
      If members(As)=false
         valid = valid and ¬ MSCk(As)
      End If
    End For
End SI2
```

Fig. 3. Algorithm SI_2 utilizing pre-computed minimal set covers

by comparing Q and MA ("$\stackrel{s}{=}$" denotes set equality). If not, one tries to extend c with each element of E, and if the new cover is minimal (tested with function *minimal*), it recursively calls MC. The minimality test verifies first if class i, to be added to partial cover c, is redundant (i.e., already covered by c). If not, it tests for each member of c if it becomes redundant when adding i to c. Tests use MA and $NewMA$, the multisets of attributes covered by c, respectively $c + i$.

In the example in Figure 1, algorithm SI_1 computes minimal covers of Q with equivalence classes of EC_q. The partial cover containing class $\{Rn, Py\}$ cannot be extended with class $\{Rn, Lu, Py\}$ because the new cover is not minimal ($\{Rn, Py\}$ becomes redundant), but can be extended with class $\{Lu\}$, which produces a minimal cover solution. The minimal covers in our example are c_{s1}=($\{Rn, Py\}$,$\{Lu\}$) and c_{s2}=($\{Rn, Lu, Py\}$). Given the sources in each equivalence class, c_{s1} produces combinations of sources (S_1, S_4) and (S_3, S_4), while c_{s2} produces only (S_2).

3.3 Algorithm SI_2

In contrast to algorithm SI_1, the novel algorithm we present in this section supposes the existence of the following structure: $MSC_k(A_s, c)$ a pre-computed relational table describing all the minimal covers of the set {1,2,...,k}. Tuple $[A_s, c]$ in MSC_k states that $A_s \in \mathcal{P}(\{1, 2, ..., k\}) - \{\emptyset\}$ is a member of minimal cover c[2]. As aforementioned[1], the number of minimal covers $|\mathcal{C}_k|$ of $\{1, ..., k\}$ increases very rapidly with k. We assume in the following that this table or its compacted N1NF version $MSC_k(A_s, C)$ where C is the list of minimal covers that include A_s as a member, holds in memory. This implies, with our implementation, an upper bound on k which is ≤ 8. Because of space limitations we do not detail the off line construction of $MSC_k(A_s, C)$: it is pre-computed, independently of any query, by an algorithm similar to algorithm SI_1.

Given MSC_k and EC_q (computed in step 1), the following relational expression calculates $\mathcal{C}_q$ the set of minimal covers of Q, which is composed only of members appearing in EC_q (equivalence classes):
$R(A_s) = \pi_{A_s}(MSC_k(A_s, c)) - \pi_{A_s}(EC_q(A_s, S))$
$S(c) = \pi_c(MSC_k(A_s, c) \bowtie R(A_s))$
$\mathcal{C}_q(c) = \pi_c(MSC_k(A_s, c)) - S(c)$

$R(A_s)$ is the set of "bad" members, where a bad member is a member of a minimal cover that is not in EC_q. Then $S(c)$ is the set of bad minimal set covers

[2] Without loss of generality, attributes names in Q are replaced by an integer in $\{1, ..., k\}$

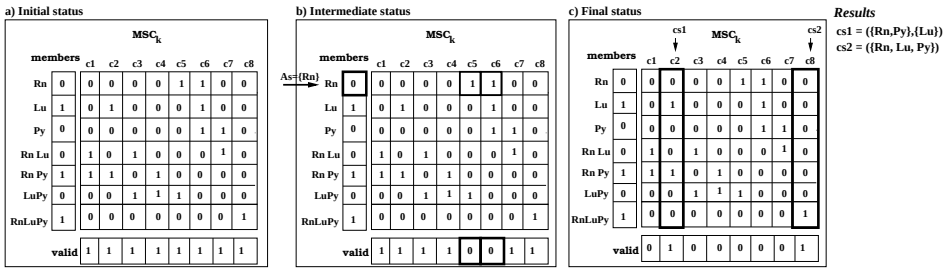

Fig. 4. SI_2 data structure

i.e. covers which hold at least one member in $R(A_s)$. Therefore $\mathcal{C}_q(c)$ is the set of "good" minimal set of covers i.e. made of members appearing in EC_q.

The algorithm in Figure 3 implements the computation of $\mathcal{C}_q(c)$. It relies on two boolean vectors:

1. *members* with size 2^k-1 where *members(A_s)* is set to 0 if the set of attributes A_s is in $R(A_s)$: it is a bad member. Note that vector *members* is constructed with no additional cost when building $EC_q(A_s, S)$ from the initial table $SA(s, A_s)$ which, recall, associates with each source its attributes.
2. *valid* with size the number of minimal set covers $|\mathcal{C}_k|$ where *valid(c)* is set to 1 if minimal set cover c is a good one: it belongs to $\mathcal{C}_q(c)$. All bits of *valid(c)* are initialized to 1.

MSC_k is implemented as a bitmap with $2^k - 1$ lines and $|\mathcal{C}_k|$ columns: $MSC_k(A_s, c)$ is set to 1 if cover c has for a member A_s.

Figure 4 displays for the example of Figure 1 the value of MSC_k and that of *members*and *valid* prior to the execution (Figure 4.a) after the first loop iteration (Figure 4.b) and at the end (Figure 4.c). With $k = 3$, there are 8 minimal covers and the final minimal covers are: $c_2 = (\{Rn, Py\}, \{Lu\})$, $c_8 = (\{Rn, Lu, Py\})$.

The size of the bitmap $|\mathcal{C}_k|$ drastically increases with k. Therefore k must be reasonably small not only because MSC_k must hold in memory but also because the complexity of the bitmap operations is $O(|\mathcal{C}_k|)$, which grows very fast. For the amount of RAM memory available in today computers, the bound is $k \leq 8$ which is reasonable for the applications we have in mind. The complexity of algorithm SI_2 is $O(2^k) \times |\mathcal{C}_k|$. However since k is small the bitmap operation is fast. This is confirmed by our experiments reported in the following section.

4 Experiments

4.1 Experimental environment

The algorithms and experiments were implemented in Java (JDK1.5.0-06), on a PC with 512 MBytes of RAM and a P4 3 GHz CPU, running Windows XP. We use a synthetic set $\mathcal{A}$ of 300 attributes and a number of sources ranging from

$N = 100$ to $N = 10000$. Assignment of attributes to sources (source indexing) follows a scheme motivated by the OpenXView model, but common to most data integration system, in which attributes are grouped in common structures (concepts in OpenXView, relations in relational models, etc). In our case, the 300 attributes correspond to 30 concepts of 10 attributes (properties) each. Source indexing and querying respect *locality*, i.e. we select attributes in several concepts instead of randomly choosing them in $\mathcal{A}$. More precisely, we used the following repartition for source indexing: 25% of the sources cover 2 concepts, 50% 3 concepts and 25% 4 concepts, with 50% of the concept's attributes (randomly chosen) in each case. Queries were randomly generated with k= 2, 4, 6 attributes randomly chosen within two concepts. The results presented in the following section are robust wrt changes in the repartitions and parameters. Because of space limitations we present only the results for the above synthetic set and queries.

4.2 Experimental results

All algorithms were run in central memory. Our performance evaluation focuses on the total time for running each of the two steps of the algorithms for source identification. Each time measure is the average over 50 random queries with the same value of k and N. We successively made three experiments: the first one evaluates step 1 which is common to both algorithms. The second one compares the performance of both algorithms for step 2 and the third experiment studies the predominance of one step over the other depending on the algorithm. Because of space limitations we only present the most illustrative figures for the general behavior of our algorithms.

Step 1 Figure 5.a displays the number of equivalence classes (number of tuples of EC_q) vs the number of sources, for queries of size $k = 6$. Some nodes are labelled by the time taken to perform step 1. The experimental results confirm that the number of classes (which theoretically can reach 2^k-1), is much smaller in practice (for $N = 4000$, a very large number of sources, the average number of classes is 33). Anyhow this number is very small wrt N for all values of N. Recall that step 2 takes as an input classes and not the sources themselves. Step 1 reduces the source identification problem to the scanning in step 2 of an expected small number of classes. In contrast, naive algorithms such as Bucket [5] compute combinations directly from the sources. As expected, the time growth is linear in N. Experiments for other values of k confirmed the same behavior and the linearity with k.

Step 2 We successively study the impact of k and of N on the time performance of step 2. Figure 5.b displays the time for performing step 2 with both algorithms vs the number of attributes k, for 4000 sources. While with SI_2 it is almost negligible for all values of k [3], with SI_1 it increases exponentially. Figure 5.c and Figure 5.d display the time for running step 2 respectively versus the number of sources (Figure 5.c) and the number of classes (Figure 5.d). They confirm

[3] Recall that for $k > 8$ the amount of memory space required by this algorithm is prohibitive

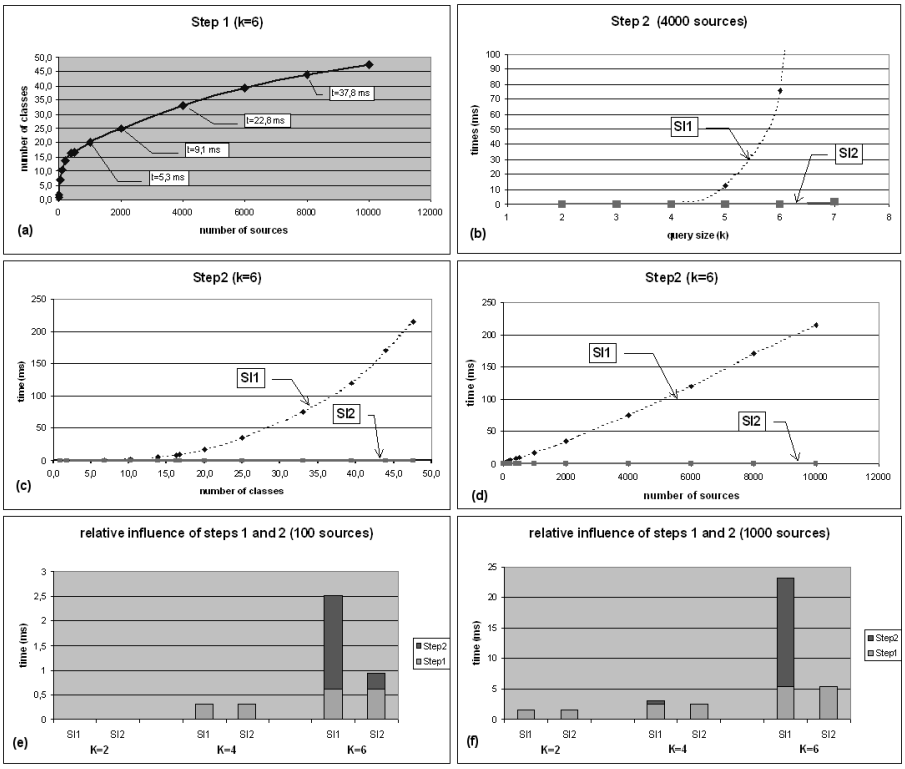

Fig. 5. Experiments

that algorithm SI_2 drastically outperforms algorithm SI_1. Time for running step 2 with algorithm SI_2 is negligible or at least less than 1 ms even for a large number of sources (classes). In contrast, algorithm SI_1 performance degrades exponentially with the number of classes (Figure 5.c). Note however the quasi-linearity with N, because of the slow growth of the number of classes with N (Figure 5.d).

Step 1 / Step 2 Last we study the relative importance of step 1 wrt that of step 2. Time versus k is plotted for 100 sources (Figure 5.e) and for 1000 sources (Figure 5.f). The first important observation is that with algorithm SI_2, when the number of sources is large, the time for running step 2 is negligible and therefore the source identification performance is that of step 1. As a matter of fact it was observed that as expected, the latter time is linear in k and N. Although we were not able to find sub-linear algorithms, step 1 still deserves some improvements. The second observation is that with algorithm SI_1, the larger k, the larger the predominance of step 2 duration over step 1, even for a relatively small number of sources: while with $k = 4$ step 2 is almost negligible, with $k = 6$, step 2 already lasts four times longer than step 1. Since step 2 duration is exponential with k, it is to be expected that with larger values of

k, the performance of source identification with algorithm SI_1 is that of step 2 which is exponential with k. Therefore, for this algorithm as well, values of k larger than 8 are unpractical.

In conclusion, we assessed the impact on performance of the organization of sources into equivalence classes depending on the query (step 1) and demonstrated the significant improvement on performance brought by algorithm SI_2 wrt to algorithm SI_1 proposed in [3]. SI_2 performs well even with a large number of sources.

5 Conclusion

This paper was devoted to source identification when a query with k attributes (keywords, properties, etc.) has to be translated into queries over a number of sources. This problem is central to query rewriting in data integration systems. We exhibited a new efficient algorithm and evaluated its performance. We showed that this algorithm is scalable and applies to environments with a very large number of sources. However this algorithm -as well as a comparable algorithm [3]- is not scalable with the number of attributes in a query. We evaluated this well known exponential behavior and gave for current technology, an experimental bound of 8 for the number of attributes in a query.

References

1. B. Amann, C. Beeri, I. Fundulaki, and M. Scholl. Querying xml sources using an ontology-based mediator. *CoopIS*, pages 429–448, 2002.
2. C. K. Baru, A. Gupta, B. Ludäscher, R. Marciano, Y. Papakonstantinou, P. Velikhov, and V. Chu. XML-Based Information Mediation with MIX. *SIGMOD*, 1999.
3. F. Boisson, M. Scholl, I. Ssebei, and D. Vodislav. Query rewriting for open xml data integration systems. *IADIS WWW/Internet*, 2006.
4. A. Deutsch, Y. Katsis, and Y. Papakonstantinou. Determining source contribution in integration systems. *PODS*, 2005.
5. A. Halvey. Answering queries using views: A survey. *The VLDB Journal*, pages 270–294, 2001.
6. V. Josifovski, P. Schwarz, L. Haas, and E. Lin. Garlic: a new flavor of federated query processing for DB2. *SIGMOD*, 2002.
7. M. Lenzerini. Data integration: a theoretical perspective. *PODS*, 2002.
8. A. Levy, A. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. *PODS*, 1995.
9. A. Macula. Covers of a finite set. *Math. Mag.*, 67:141–144, 1994.
10. R. Pottinger and A. Halevey. Minicon: A scalable algorithm for answering queries using views. *The VLDB Journal*, pages 182–198, 2001.
11. D. Vodislav, S. Cluet, G. Corona, and I. Sebei. Views for simplifying access to heterogeneous XML data. *CoopIS*, 2006.
12. C. Yu and L. Popa. Constraint-based xml query rewriting for data integration. *SIGMOD*, 2004.