Data Integration by Describing Sources with Constraint Databases

Xun Cheng[1,*] Guozhu Dong[2,**] Tzekwan Lau[1] Jianwen Su[1,*]
xun@cs.ucsb.edu gdong@cs.wright.edu tzekwan@cs.ucsb.edu su@cs.ucsb.edu

[1] Department of Computer Science, University of California, Santa Barbara, CA 93106, USA
[2] Department of Computer Science and Engineering, Wright State University, Dayton, Ohio 45435, USA

Abstract

We develop a data integration approach for the efficient evaluation of queries over autonomous source databases. The approach is based on some novel applications and extensions of constraint databases techniques. We assume the existence of a global database schema. The contents of each data source are described using a set of constraint tuples over the global schema; each such tuple indicates possible contributions from the source. The "source description catalog" (SDC) of a global relation consists of its associated constraint tuples. Such a way of description is advantageous since it is flexible to add new sources and to modify existing ones. In our framework, to evaluate a conjunctive query over the global schema, a plan generator first identifies relevant data sources by "evaluating" the query against the SDCs using techniques of constraint query evaluation; it then formulates an evaluation plan, consisting of some specialized queries over different paths. The evaluation of a query associated with a path is done by a sequence of partial evaluations at data sources along the path, similar to side-ways information passing of Datalog; the partially evaluated queries travel along their associated paths. Our SDC-based query planning is efficient since it avoids the NP-complete query rewriting process. We can achieve further optimization using techniques such as emptiness test.

1. Introduction

When a collection of autonomous data sources are to work together in a loosely coupled setting, serious problems arise [22, 26, 18, 29, 13]. Examples of such applications include digital libraries [4] and electronic commerce [6, 7]. Two of the fundamental issues in developing such data integration applications are: (1) managing semantic heterogeneity (schema integration) and (2) efficiently evaluating queries that are often distributed in nature (query optimization).

Among current approaches to data integration, data warehousing integrates relevant data sources into a repository that can be fully materialized [12], fully virtual, or a combination [14]. Queries are formulated against the integrated schema at the data warehouse and are evaluated either locally at the data warehouse when it is fully materialized or decomposed into subqueries that are evaluated at the sources when some or all of the data are not materialized. Wrappers and mediators [30, 8, 25, 28] present an alternative. Intuitively, a semantic wrapper of a data source exports both the structure and query capabilities of the source. When a query is received, a mediator will process the query (e.g., decomposition and result merging) by requesting necessary input from the wrappers available to it. Unlike the case of data warehousing where the integration is mostly "hardwired", as long as the wrappers for new sources are developed and properly introduced to the mediators, the new sources can be added but this may cause significant changes to mediators. The agent-based approach [17, 3, 16, 9] uses simple formal languages such as description logics (see [13]) to wrap data sources in terms of some global model (ontology). To evaluate a query, an agent determines the sources capable of answering (a part of) the query using the techniques of rewriting (conjunctive) queries with (conjunctive) views [20, 24, 16]. The advantage of using such information agents is that it is very easy to add new sources since the language for defining wrapper is highly conceptual and simple. However, the inherent difficulty (NP-complete) to query rewriting using views prohibits efficient implementation of information agents.

In this paper we propose a data integration framework for the efficient evaluation of queries over multiple autonomous data sources (databases). Our approach combines both the wrappers/mediator and information agent approaches. The structural integration assumes a global schema but uses wrappers and lightweight mediators to represent source databases. In principle, the contents of each data source are described using a set of constraint tuples over the global schema; each such tuple indicates possible contributions from the wrapped source. In this, we only allow selections and projections, rather than arbitrary conjunctive views, in describing the contents of data sources. In addition, we organize the source descriptions into a constraint database [15], called "source description catalogs" or "SDCs", and then extend the constraint query evaluation techniques [10] and constraint solving techniques [11] in constructing query evaluation plans. As a result, we not only retain the flexibility of adding new sources as in information agents, but also are able to provide efficient (polynomial time) query evaluation algorithms.

We also develop a scalable framework of evaluating queries, assuming each source provides simple querying capabilities. Our approach is applicable when sources can only do selections and projections, and is also applicable and perhaps more efficient when the source wrappers can do semi-joins. Using constraint-based SDCs, a plan generator decomposes a query into many "evaluation paths" through which some constrained copies of the query will travel in parallel. For each such path, the associated query

* Supported in part by NSF grant IRI-9411330 and IRI-9700370.
** Supported in part by a grant from the Australian Research Council while at the University of Melbourne. Part of work done while visiting UCSB.

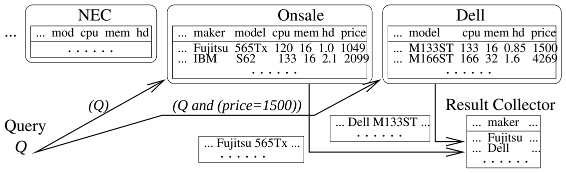

Figure 1. Evaluation of Q

moves from one source to the next in a sequential fashion, and each site performs a partial evaluation (either (*) selection and projection with iteration or (**) with semi-join) to contribute what it can towards the answer. We also consider query optimization issues aiming at reducing the amount of data to be transferred between sources. In particular, we study both *static* and *dynamic* optimization techniques. For static optimization, we use SDCs to tighten selection conditions, and order subqueries to "push selections down". We also consider the optimization of the SDC before generating the evaluation plans, in order to reduce communication cost. For the dynamic part, we prune evaluation paths that will not contribute to the answers, tighten selection conditions after partial evaluations, and also consider sharing of common prefix of many evaluation paths.

Our approach of "cataloging" data sources by (1) structural wrappers with lightweight mediators and (2) view-like descriptions of wrapped sources provides a modularized method to data integration, it is extensible (allow changes to sources) and allows tractable query processing. It is also practical since the wrappers and source descriptions are very easy to build and query evaluation is based on exsiting techniques and generic.

Our distributed evaluation approach is related to but different from Mariposa [27], where query evaluation is based on decomposing a query into subqueries which are then posted for data sources to bid, and different from the ordering of semi-joins in mediators [1].

We illustrate the main ideas with a motivating example in §2. §3 introduces the basic notions including data sources, and queries over a collection of data sources, while §4 defines the source description model. §5 and §6 discuss how to construct evaluation plans for queries. §7 presents a basic algorithm for (distributed) evaluation of queries, and §8 includes a discussion on optimization and improvements over the basic algorithm. §9 discusses future work.

2. A Motivating Example

We illustrate the main ideas of our approach with a simplified example concerning online information and sales of personal computer systems. Existing on the web are the following services: (1) online catalogs containing the configuration and pricing information of computer systems from vendors or manufacturers, (2) reviews of systems/vendors.

Currently, shoppers who use these information services need to deal with the semantic differences of data and interfaces, and often find unsatisfactory answers. Some existing integrated services (e.g., www.uvision.com) copy periodically everything from the sources; such an approach not only is unscalable but also incurs high maintenance cost. It is unsuitable for frequently changed catalogs.

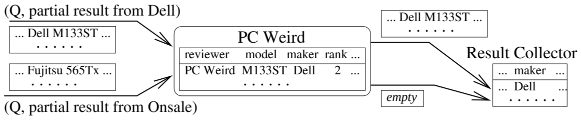

Figure 2. Partial evaluation of Q' at sources

In our framework, we first design a global relation schema $Sys(v, mk, mod, cpu, mem, p)$ representing the information about *v*endor, *mak*er, *mod*el, *cpu* speed (in MHz), *mem*ory size (in MB), and *p*rice of computer systems. The individual data sources are described in a constraint relation [15] such as the following.

vendor	maker	model	cpu	mem	price	*site*
$\perp$	NEC	$\top$	$\geqslant 75$	$[4, 128]$	$\perp$	NEC
Dell	Dell	$\top$	$\geqslant 90$	$[8, 128]$	$\geqslant 1500$	DELL
Onsale	$\top$	$\top$	$\top$	$\top$	$\top$	ONSALE
...						

Here "$\top, \perp$" are two special symbols which indicate if the site has (does not have) data about the corresponding attribute. Such a constraint relation is called a "source description catalog." A tuple t in this catalog is used to indicate that the data source may contain relational tuples (for those non-$\perp$ columns of t) satisfying this t as a constraint; and the catalog is used to indicate that each relational tuple at a data source should satisfy at least one of the catalog tuples about the data source.

Consider a query Q that lists all systems that have at least a 100 MHz cpu, 16 MB memory, and cost $\leqslant \$1,500$. By evaluating Q against the source description catalog (which tells us among other things that the NEC site has no price information), we found that the answer to the query is the union of at least two subqueries (Fig. 1): one (modified from Q by further requiring price = 1500) is sent to DELL, and another (Q itself) is sent to ONSALE.

Suppose we have a relation for reviews *Rev*(*reviewer, mod, mk, rank, yr*). Consider the query Q': find the same systems as specified by Q that are ranked among the top 20 by PC Weird. By evaluating Q' against the SDC, we found that the answer to the query is the union of at least two subqueries (Fig. 2): one to join the information at DELL with that at PC Weird with some selection conditions; another to join the information at ONSALE with that at PC Weird.

3. Conjunctive Queries

We assume standard notions of relational databases including relation/database schemas/instances [2] and also allow domains with a dense or discrete order $<$. A *conjunctive query* Q is an expression of the form:

$$ans(\bar{x}) \leftarrow r_1(\bar{y}_1), ..., r_k(\bar{y}_k), \varphi$$

where each r_i is a relation name, $\bar{x}, \bar{y}_i$'s are sequences of (not necessarily distinct) typed variables, and φ is a conjunctive formula consisting of equality and order comparisons, among constants and variables in $\bar{x}, \bar{y}_i$, satisfying typing constraints. Each $r_i(\bar{y}_i)$ is also called a *subgoal* of Q. Without loss of generality, we assume throughout this paper that the variables in $r_i(\bar{y}_i)$ part are always distinct, since equalities can be pushed into the formula φ. A variable in Q is *essential* if it occurs in the head, or is involved in a predicate in φ with a constant or a variable other than itself.

Example 3.1 Consider the query Q' in the online shopping example in §2. Q' can be expressed as:

$$ans(x_1, x_3, x_4, x_6) \leftarrow Sys(x_1, ..., x_6), Rev(x_7, ..., x_{11}), x_2{=}x_9, x_3{=}x_8, 100{\leqslant}x_4, 16{\leqslant}x_5, x_6{\leqslant}1500, x_7{=}\text{“PC\_Weird”}, x_{10}{\leqslant}20.$$

All but x_{11} are essential variables. ∎

In the context of data integration, each data source needs to have a “semantic wrapper” [13] so that both the information contents and the query capabilities can be offered to the outside. It is important to have a scalable integrated query evaluation framework, in addition to schema integration. The work in this paper is a step towards this goal.

We assume that a global schema is available and the collection of data from all data sources represents a (possibly incomplete) database over the global schema. Specifically, for each global relation r, a data source α may contain a subset of $\pi_X(r)$, where X are some attributes of r. When it is clear, we denote the subset as $\pi_\alpha r$.

Example 3.2 Consider again the global relation schema $Sys(v, mk, mod, cpu, mem, p)$ in the example in §2. Since NEC is not a direct vendor, the data source at NEC may contain a subset $\pi_{\text{NEC}} Sys \subseteq \pi_{mk,\, mod,\, cpu,\, mem} Sys$. ∎

In Information Manifold a data source may contain a subset of a conjunctive view over the global database. In this sense our model of data sources is a simplified version of Information Manifold. However, as we will describe in the next section, we use constraint database techniques in representing the content descriptions instead of containment rules. Consequently, we are able to develop a scalable query evaluation framework and optimization techniques.

Let r be a relation schema. If t is a tuple in some data source and t contains values for only attributes in S where S is a proper subset of r, then we view t as a *generalized tuple* (in the sense of constraint databases) over r where all other attributes are existentially quantified: $\tilde{t} = \exists A_1 \cdots \exists A_k t$ with $A_1, ..., A_k$ being all attributes in r but not in S. Therefore, each data source contains a set of generalized tuples over the global schema. The union $\tilde{d}$ of the sets of generalized tuples at all data sources is called a *generalized database*. A generalized database is an incomplete database and represents a set of possible databases.

Given a conjunctive query Q over the global schema D and a generalized database $\tilde{d}$, the *definite answer* of Q in $\tilde{d}$ is the intersection of all answers of Q in all databases $\tilde{d}$ represents. In other words, the definite answer of a query is the logic consequence of Q and $\tilde{d}$. Notice that the definite answer of a query is a set of tuples (not generalized tuples). In the remainder of the paper, we focus on definite answers of conjunctive queries and we will use the word “answers” for the phrase “definite answers”.

4. Source Description Catalogs (SDCs)

In our approach to describing data sources, we assume the existence of a global database schema (also called an “ontology” in the literature), similar to the information-agent based approaches [17, 21, 3, 16, 9]. The contents of the autonomous data sources are described using constraints in terms of the global schema. The relevant information from a data source, for each relation in the global schema, is described as quantifier-free (first-order) formulas using equality and order (dense or discrete) predicates; the set of all such descriptions will be called a “source description catalog” (SDC) for the given relation. An SDC is actually a “constraint relation” in the sense of [15] over dense [10] or discrete order domains.

Our constraint-database description of data sources is more advantageous, by allowing flexible addition and modification of sources, than approaches such as Tsimmis [8] where the correspondence between sources and the global schema is compiled into the mediators. As we shall see later, our SDC-based query planning is more efficient than approaches such as Information Manifold [17] by avoiding their NP-complete query rewriting process. Furthermore, we also achieve some query-evaluation time advantages: The evaluation is completely distributed; there is no need for processing after receiving the answers except collecting them; and the data being transmitted from sites to sites are much tighter and more relevant to answering the query.

Let $\top, \bot$ be two new symbols not in any domains, A an attribute, and τ a domain. The *descriptive values* (*d-values*) of $A{:}\tau$ are defined as follows. (1) $A{=}\top$ and $A{=}\bot$ are d-values, (2) $A\theta c$ is a d-value if θ is an order/equality predicate on the domain τ and c a value in τ, (3) $\phi \wedge \psi$, $\phi \vee \psi$, $\neg\phi$ are d-values if ϕ, ψ are d-values that do not involve $\bot, \top$.

Each data source is identified by an unique *address* of the source. Let $R = (A_1{:}\tau_1, ..., A_n{:}\tau_n)$ be a relation schema. A *descriptive tuple* (*d-tuple*) of R is a tuple $[v_1, ..., v_n, \alpha]$ where each v_i is a d-value of $A_i{:}\tau_i$ and α an address. (This notion is generalized to include many addresses in the next section.) A *source description catalog* (*SDC*) of the relation R, denoted by $\mathcal{C}_R$, is a finite set of d-tuples of R.

Intuitively, a d-tuple t about the data source α indicates that α may contain (relational) tuples (for those non-$\bot$ columns of t) satisfying t as a constraint. The set of d-tuples in $\mathcal{C}_R$ about α indicates that each tuple at α should satisfy at least one of these d-tuples.

Example 4.1 Consider the *Sys* relation in the online shopping example of §2. Suppose α_{N} is the address for NEC. The SDC of *Sys* contains the following d-tuple t_1:

$[v{=}\perp, mk{=}\text{NEC}, mod{=}\top, 75{\leqslant}cpu, 4{\leqslant}mem{\leqslant}128, p{=}\perp, \alpha_{\text{N}}]$

For simplicity, we abbreviate t_1 as $[\perp, \text{NEC}, \top, [75, \infty), [4, 128], \perp, \alpha_{\text{N}}]$. The d-tuple t_1 indicates that the data source α_{N} may contain tuples of form (NEC, x, y, z) where $75 \leqslant y$ and $4 \leqslant z \leqslant 128$. Observe that α_{N} does not contain values for the attributes marked with $\perp$ in t_1. ∎

Our approach to source descriptions is different from those of [17, 21, 3, 16, 9], where the contents of a data source are described as conjunctive views. Our SDCs can clearly be viewed as descriptions of the query processing capabilities of wrappers (e.g. [8]).

5. Locating Sources

An algorithm is given in this section to locate combinations of data sources that may contribute to the user query; the generation of an evaluation plan and the evaluation algorithm are discussed in §6 and §7.

To generate an evaluation plan for a conjunctive query Q over the global schema, we first evaluate a constraint database query ([15]) Q_α against the SDCs of the involved relations. The answer to Q_α gives a set of sequences of data source addresses, which may contribute to the answer of Q. For each such sequence, we will (1) perform a global optimization to narrow the selection conditions in the query, (2) order the subgoals of the query, and (3) construct a query evaluation plan (steps (2) and (3) will be described in §6).

The evaluation of Q_α is similar to but generalizes that for dense-order constraint queries [10] due to the presence of discrete orders. To describe the algorithm for locating data sources, we use the standard notion of an interval. We will need to manipulate the lower and upper bounds of intervals that can be open or closed. We use notations like $(_a$, $[_a$ for lower bounds and $_a)$, $_a]$ for upper bounds with the obvious meanings. We allow comparisons between lower bounds and between upper bounds; the truth value of such a comparison is determined by: $[_a$ and $(_a$ are smaller than $[_b$ and $(_b$ for all $a < b$, $[_a$ is smaller than $(_a$, and $_a)$ is smaller than $_a]$. If l (u) is a lower (upper) bound, we define:

$$up(l) = \begin{cases} (_a & \text{if } l = (_a \text{ and the domain order is dense, or } l = [_a \\ (_{a+1} & \text{if } l = (_a \text{ and the domain order is discrete} \end{cases}$$

$$down(u) = \begin{cases} _a) & \text{if } u{=}_a) \text{ and the domain order is dense, or } u{=}_a] \\ _a) & \text{if } u{=}_{a+1}) \text{ and the domain order is discrete} \end{cases}$$

where $a{+}1$ denotes the successor of a for a discrete order.

A *condition* over distinct variables $x_1, ..., x_n$ is a tuple $C = (v_1, ..., v_n, (\theta_{ij}))$ where each v_i is an interval limiting the domain of x_i and (θ_{ij}) is an $n{\times}n$ matrix where θ_{ij} is in $\{=, \leqslant, <, \geqslant, >, ?\}$ restricting the relationships between the variables. Clearly, C can be expressed by a formula $\Phi(C)$.

Let φ be a conjunctive formula over (distinct) variables $x_1, ..., x_n$ involving predicates $=, \leqslant, <$. A condition C is said to be a *representation* of φ if $\Phi(C)$ is logically equivalent to φ. A representation C of φ is *canonical* if the intervals in C are tight, and whenever two variables x_i, x_j are connected by a sequence of variable comparisons in φ, the symbol θ_{ij} in C is also tight.

A canonical representation of a conjunctive formula φ over $x_1, ..., x_n$ is constructed in two steps. For the matrix, we first construct an edge-labeled directed graph G_φ with nodes $x_1, ..., x_n$ and edges (x_i, x_j) with label $\leqslant$ (or $<$) if $x_i \leqslant x_j$ (resp. $x_i < x_j$) is in φ. If G_φ has a cycle including an edge labeled $<$, φ is unsatisfiable and the query answer is empty. Otherwise, we find the strongly connected components (SCCs) of G_φ. For each pair of distinct nodes x_i, x_j, if they are in the same SCC, let θ_{ij} be "$=$"; if there is a path from x_i to x_j we let θ_{ij} be "$<$" if an edge on some path from x_i to x_j is labeled "$<$", and "$\leqslant$" otherwise.

To obtain the intervals, let $L_i{=}(_{-\infty}$ and $U_i{=}_{+\infty})$ be the initial lower and upper bounds (resp.) for each $1{\leqslant}i{\leqslant}n$. We improve the bounds using comparisons between variables and constants. For example, if the comparison is "$c{<}x_i$", let $L_i = \max\{L_i, (_c\}$, if it is "$x_i < c$" let $U_i = \min\{U_i, {_c})\}$. We then use G_φ to propagate lower bounds upward and upper bounds downward:

1. For each SCC S of G and each $x_i \in S$, let $L_i = \max\{L_j \mid x_j \in S\}$ and $U_i = \min\{U_j \mid x_j \in S\}$;
2. If G has an edge (x_i, x_j) labeled "$\leqslant$", let $L_j = \max\{L_i, L_j\}$ and $U_i = \min\{U_i, U_j\}$; and
3. If G has an edge (x_i, x_j) labeled "$<$", let $L_j = \max\{up(L_i), L_j\}$ and $U_i = \min\{U_i, down(U_j)\}$.

If any pair L_i, U_i defines an empty interval, φ is not satisfiable. Otherwise, let B_φ be the set of variables x_i such that $L_i \neq (_{-\infty}$ or $U_i \neq _{+\infty})$.

The construction of the intervals in the canonical representation is similar to an algorithm in [11] and can be done in linear time in the length of the formula.

Example 5.1 Consider the formula $\varphi = 3 < x_1 \leqslant x_2 < x_3 \wedge x_2 \leqslant 9 \wedge x_4 < x_5 < 6$ where x_1, x_2, x_3 are **real** variables and x_4, x_5 are **int** variables. A canonical representation of the formula is $C =$

$$\left((3, 9], (3, 9], (3, +\infty), (-\infty, 5), (-\infty, 6), \begin{bmatrix} = & \leqslant & < & ? & ? \\ \geqslant & = & < & ? & ? \\ > & > & = & ? & ? \\ ? & ? & ? & = & < \\ ? & ? & ? & > & = \end{bmatrix} \right)$$ ∎

Let $Q : ans(\bar{x}) \leftarrow r_1(\bar{y}_1), ..., r_k(\bar{y}_k), \varphi$ be a satisfiable conjunctive query with essential variables $x_1, ..., x_n$. Suppose $C{=}(v_1, ..., v_n, (\theta_{ij}))$ of φ is the canonical representation of φ. We evaluate the following constraint query

$sites(\alpha_1, ..., \alpha_k) \leftarrow \mathcal{C}_{r_1}(\bar{x}_1, \alpha_1), ..., \mathcal{C}_{r_k}(\bar{x}_k, \alpha_k), \varphi$

against the SDCs of the database as follows. We compute the join $\mathcal{C} = \mathcal{C}_{r_1} \bowtie \cdots \bowtie \mathcal{C}_{r_k}$ (with proper attribute renaming) where $\perp$ is not equal to anything. For each d-tuple t in $\mathcal{C}$ and each attribute A, if $t.A$ is $\perp$ and the corresponding variable in Q is essential, t is removed. Otherwise Algorithm 1 given below checks satisfiability of the conjunction of t and C, and if satisfiable, produces (1) a set of selection conditions (intervals) for each variable and (2) boundness information of variables.

Since SDCs are constraint relations, possibly with orders, each column of a SDC is a set of intervals. We can build indexes on the attributes of SDCs and apply efficient join algorithms for constraint relations [31].

Let $A_1, ..., A_n$ be the attributes for the essential variables in Q. Then $t.A_i \neq \perp$ for each A_i. Since each d-value $t.A_i$ is quantifier-free, $t.A_i$ defines a finite set of intervals. For a given pair t and $C = (v_1, ..., v_n, (\theta_{ij}))$, Algorithm 1 will output $(I_1, ..., I_n, B)$, where each I_i is a set of (tight) intervals for the variable x_i and $B \subseteq \{x_1, ..., x_n\}$ is a set of bound variables (i.e., having selection conditions).

Algorithm 1: Generation of Sites and Conditions
Step 1: Let $B = \emptyset$ and I_i be the set of intervals represented by the d-value $t.A_i$ for each i.
Step 2: For each i let $I_i := I_i \cap v_i$. If $I_i = \emptyset$ for some i, $t \wedge \varphi$ is not satisfiable and no site sequence is produced.
Step 3: We iteratively modify the I_i's until no changes can be made. Suppose L_i, U_i, L_j, U_j are the least lower and greatest upper bounds of all intervals in I_i, I_j (resp.). The modification rules depend on θ_{ij} being
(=) Let $I_i := I_j := I_i \cap I_j$.
($\leqslant$) Let[1] $I_i := I_i \cap (-\infty, U_j)$ and $I_j := I_j \cap (L_i, +\infty)$.
(<) Let $I_i{:=}I_i{\cap}(-\infty, down(U_j))$ and $I_j{:=}I_j{\cap}(up(L_i), +\infty)$.
($\geqslant$) Let $I_i := I_i \cap (L_j, +\infty)$ and $I_j := I_j \cap (-\infty, U_i)$.
(>) Let $I_i{:=}I_i{\cap}(up(L_j), +\infty)$ and $I_j{:=}I_j{\cap}(-\infty, down(U_i))$.
Step 4: Finally we add to B each x_i where the value of I_i has been changed and each x_i in B_φ.

Example 5.2 Consider the canonical representation C in Example 5.1. Initially let $I_1{=}\{(0,3), (4.5, 5], [7, 8)\}$, $I_2{=}\{(3.5, +\infty)\}$, $I_3{=}\{[4, 7.5)\}$, $I_4{=}\{[-5, 0), (1, 10]\}$, $I_5{=}\{(-\infty, -10], [-7, 3]\}$. Step 2 of Algorithm 1 results in $I_1{:=}I_1{\cap}v_1 =\{(4.5, 5], [7, 8)\}$, $I_2{:=}I_2{\cap}v_2{=}\{(3.5, 9]\}$, $I_3{:=}I_3{\cap}v_3{=}\{[4, 7.5)\}$, $I_4{:=}I_4{\cap}v_4{=}\{[-5, 0), (1, 4]\}$, $I_5{:=}I_5{\cap}v_5{=}\{(-\infty, -10], [-7, 3]\}$. Step 3 further tightens the I_i's. For example, since $x_1{\leqslant}x_2$, we set I_1 as $I_1{\cap}(-\infty, U_2)$ and set I_2 as $I_2{\cap}(L_1, +\infty)$, where $U_2{=}\ _9]$ is the greatest upper bound for I_2, and $L_1{=}\ (_{4.5}$ is the least lower bound for I_1. This results in a new value $I_2{=}\{(4.5, 9]\}$. For integer variables (discrete order domain), bound propagation is different. For $x_4{<}x_5$, we get $I_4{:=} I_4{\cap}(-\infty, down(U_5)){=}\{[-5, 0), (1, 2]\}$ where $U_5 = \ _3]$ and *down*($_3]$) $=\ _2]$ since the order is discrete. The final values of I_i's are $I_1 = \{(4.5, 5], [7, 7.5)\}$, $I_2 = \{(4.5, 7.5)\}$, $I_3 = \{(4.5, 7.5)\}$, $I_4 = \{[-5, 0), (1, 2]\}$, $I_5 = \{[-4, 3]\}$. ▮

6. Subgoal Ordering and Plan Generation

In this section we consider the generation of efficient query evaluation plans. The main task is to (re-)order data sources to reduce the sizes of temporary results. More general optimization techniques will be the focus of §8.

As illustrated in §2, a query over the global schema will be partially evaluated at the relevant sites, and the temporary (or final) results will be sent to the subsequent sites. Similar to the processing of joins in central or distributed databases, the size of the temporary results is a dominant factor in complexity for our situation: larger results would require more communication time and may lead to longer evaluation time at the next site. The size of temporary results is dependent on the ordering of subqueries.

Example 6.1 Continuing with the online shopping example in §2, consider the query Q: find systems ranked among the top 5 in the most recent PC Weird ranking:

$$ans(x_1, x_2, ..., x_7) \leftarrow Sys(x_1, ..., x_6), Rev(x_7, ..., x_{11}),$$
$$x_2{=}x_9, x_3{=}x_8, x_{10} \leqslant 5, x_7{=}\text{“PC Weird”}, x_{11}{=}1998.$$

One way to answer the query is to join *Sys* at Onsale with *Rev* at PC Weird. One can start the query at Onsale, and send the result to PC Weird for further evaluation, or vice versa. Suppose that there are 100 tuples in *Sys* at Onsale. If we first evaluate the query at Onsale, we have to send 100 tuples to PC Weird. On the other hand, if we first evaluate Q at PC Weird, we only need to send 5 tuples to Onsale. ▮

The subgoal ordering problem for our situation is different from the traditional central or distributed processing environment, since the systems are autonomous and indexing (information) is generally not available. As we will discuss in the next section, the query evaluation is performed by iterating through tuples in a temporary relation received by a site, performing a selection and projection, and sending the result to the next site on a path. Clearly the smaller the sizes of the intermediate relations are, the more efficient the evaluation process will be. The ordering problem resembles sideways information passing in Datalog optimization and it can be addressed with the adornment technique [2].

Let Q be a conjunctive query involving k subgoals $r_1, ..., r_k$ and with n essential variables $x_1, ..., x_n$. For each d-tuple t in the join $\mathcal{C}_{r_1}{\bowtie}...{\bowtie}\mathcal{C}_{r_k}$, we use Algorithm 1 to determine if t and Q are consistent and, if they are, to generate the set B of bound variables and the variable comparison matrix (θ_{ij}). Algorithm 2 below derives an ordering f, the schemas of temporary relations s_i ($1 \leqslant i \leqslant k$), and sets of bound variables b_i ($1 \leqslant i \leqslant k$) for subgoals (so that selection conditions can be constructed during evaluation).

Algorithm 2: Subgoal Ordering
Step 1: Let $g = \{r_1, ..., r_k\}$.
Step 2: For $i := 1..k$ do:
2.1: We select (and remove) r_j from g such that r_j has the fewest number of unbound essential variables. Let $f(i){=}j$.
2.2: Let s_i be the set of essential variables x such that x is in s_ℓ for some $\ell \leqslant i$ and x appears in the goal (answer) or is compared (in θ_{ij}) with some variable in a subgoal in g Let b_i be the set of bound variables in r_j.

[1]The last “)” in “$(-\infty, U_j)$” is added for clarity reasons.

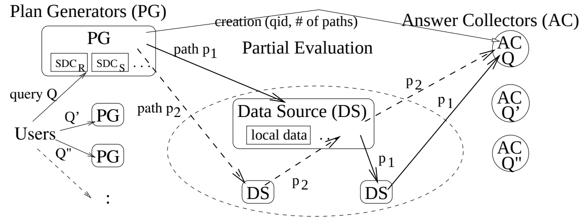

Figure 3. Queries travel along paths

Add to B (1) all essential unbound variables occurring in r_j, (2) all essential variables that are equal to some variables in B according to the comparison matrix θ_{ij}.

Step 3: Output the function f, schemas $s_1, ..., s_k$, and variable sets $b_1, ..., b_k$.

Example 6.2 Consider the following conjunctive query

$$ans(x_1, x_8, x_9) \leftarrow r_1(x_1, x_2, x_3), r_2(x_4, x_5, x_6), r_3(x_7, x_8, x_9), x_3{=}x_5, x_6{=}x_7, x_2{=}a, x_4{=}b.$$

All variables are essential. Let $B{=}\{x_2, x_4\}$ be the initial set of bound variables. The first step of the ordering is to compute the bound-free conditions of each essential variable: $r_1^{\mathrm{fbf}}, r_2^{\mathrm{bff}}$, and r_3^{fff}. Since both r_1, r_2 have 2 unbound variables, r_2 is chosen at random. Then, x_5, x_6 and consequently x_3, x_7 become bound and the adornments are r_1^{fbb} and r_3^{bff}. Now r_1 is favored since it has fewer unbound essential variables. Thus $f(1){=}2, f(2){=}1$ and $f(3) = 3$. Also, $b_1 = \{x_4\}, s_1 = \{x_5, x_6\}, b_2 = \{x_2, x_3\}, s_2 = \{x_1, x_6\}, b_3 = \{x_7\}, s_3 = \{x_1, x_8, x_9\}$. ∎

An *evaluation path* for the d-tuple t of the form $p_t = (\alpha_{f(1)}, r_{f(1)}, b_1, s_1, ..., \alpha_{f(k)}, r_{f(k)}, b_k, s_k, I_1, ..., I_n, \theta_{ij})$ can now be produced, where α_i is the address for the relation r_i, I_i's are the interval sets for the essential variables, (θ_{ij}) is the matrix in a canonical representation of the formula in Q, f is the ordering function, s_i's are schemas, and b_i's are bound variables sets.

Example 6.3 Consider the query Q' in Example 3.1. The following is an evaluation path t:

$$(\alpha_{\mathrm{P}}, Rev(x_7, ..., x_{11}), \{x_{10}\}, s_1(x_8, x_9),\ \alpha_{\mathrm{O}}, Sys(x_1, ..., x_6), \{x_2, x_3, x_4, x_5, x_6\}, s_2(x_1, x_3, x_4, x_6), I_1, ..., I_{10}, (\theta_{ij}))$$

where $I_1{=}\{\text{Onsale}\}$, $I_2{=}I_3{=}I_8{=}I_9{=}\{(-\infty, +\infty)\}$, $I_4{=}\{[100, +\infty)\}$, $I_5{=}\{[16, +\infty)\}$, $I_6{=}\{(-\infty, 1500]\}$, $I_7{=}\{\text{PC\_Weird}\}$, $I_{10}{=}\{(-\infty, 20]\}$, and (θ_{ij}) represents $x_2{=}x_9 \wedge x_3{=}x_8$. ∎

Suppose $t_1, ..., t_\ell$ are all d-tuples in SDCs that are satisfiable with Q, the evaluation plan for Q is a set of evaluation paths $\{p_{t_1}, ..., p_{t_\ell}\}$.

Theorem 6.4 (1) For each evaluation path, the selection conditions (or the interval sets) for essential variables are tight. (2) The evaluation plan for Q can be generated in polynomial time in the size of SDCs.

7. Partial Evaluation

For each query Q, the *plan generator* constructs the evaluation plan of Q using Algorithms 1 and 2 and creates an *(answer) collector* for Q (Fig. 3). The plan generator also constructs a "path envelope" (described below), for each path in the plan, which is then sent to the first site in the path. When a source receives a path envelope, it constructs a local query from the envelope and evaluates the query. If the result is nonempty and there are more sites in the path, it composes a new path envelope and sends it to the next site. Otherwise, the result to the local query is sent to the collector. When the collector has received the results from all paths, the entire evaluation of the query is completed.

The evaluation strategy developed in this paper differs from those mediator-based strategies [8, 25] or agent-based ones [17, 21, 3, 16]. In these cases, the mediator or agent plays a greater role: it not only decomposes the query into subqueries (e.g., subgoals) and sends them to data sources, but also needs to perform further manipulations on the results from the sources to produce the final answer. Our approach distributes the task of result merging to the data sources and there is no "central controller" for merging results. The entire process of query evaluation is in polynomial time in the sizes of databases and SDCs. Although this requires slightly more query processing ability at data sources, it can be easily provided by a thin wrapper.

A path envelope contains (1) a partially evaluated query with ordered subgoals, (2) a temporary relation (the result of completed partial evaluations to be used), and (3) conditions for selections and joins. A *path envelope* of a query Q has the following general form (qid, α_a, s_0, t), where *qid* is the unique identifier of Q, α_a the address of the answer collector, s_0 a (temporary) relation, and t some suffix of an evaluation path of Q.

When a path envelope (qid, α_a, s_0, t) reaches a source α where $t = (\alpha, r, b, s, ..., I_1, ..., I_n, (\theta_{ij}))$ is a suffix of an evaluation path, the current source constructs a local query[2] $\pi_X(\sigma_F(s_0 \bowtie \pi_\alpha r))$, where X is the set of all attributes in s, $\pi_\alpha r$ is the local part of the relation r, and F a selection condition obtained from $I_1, ..., I_n, (\theta_{ij})$, and the bound variables in b. If the answer s' to the local query is empty, the evaluation path will definitely not contribute any answer tuple, and an empty set is sent to the collector α_a. Otherwise, if there is a next source α', a new path envelope (qid, α_a, s', t') is sent to α', where t' is the result of deleting α, r, b, s from t; otherwise, s' is sent to α_a.

Algorithm 3: Partial Evaluation

Let the input path envelope be (qid, α_a, s_0, t) where $t = (\alpha, r, b, s, ..., I_1, ..., I_n, (\theta_{ij}))$.

Step 1: Compose a query $E = \pi_X \sigma_F(s_0 \bowtie \pi_\alpha r)$, where $\pi_\alpha r$ is the local relation for r, X the set of all attributes in s, and F constructed such that, whenever x_i, x_j are essential and occurring in s_0 or r (either x_i or x_j must occur in r),

- if $\theta_{ij} \neq$ "?", F includes the condition $x_i \theta_{ij} x_j$, and
- if x_i is in b, F has the selection for I_i.

Step 2: Evaluate E locally and get the answer s'.

[2] $(s_0 \bowtie \pi_\alpha r)$ is similar to a semi-join; if the source is incapable of performing a join, a thin wrapper to iterate through tuples in s_0 is needed.

Step 3: If $s' = \emptyset$ or α is the last, send s' to α_a and stop.

Step 4: Otherwise, let α' be the next source and send the new path envelope (qid, α_a, s', t') to α', where t' is obtained from t by deleting α, r, b, s.

When a plan generator constructs the first path envelope, the temporary relation has no attributes but is nonempty (so that the join at the first source will not be always empty).

Example 7.1 The path envelope for the path t in Example 6.3 is $p{=}(qid, \alpha_a, r_{true}, t)$, where r_{true} is the nonempty relation with no attributes. We show how Algorithm 3 works on p. Initially p is sent to $\alpha_{\rm P}$. After the evaluation at $\alpha_{\rm P}$, a new envelope $p_1{=}(qid, \alpha_a, s_1(x_8, x_9), t_1)$ arrives at $\alpha_{\rm O}$, where $t_1{=}(\alpha_{\rm O}, Sys(x_1, ..., x_6), \{x_2, ..., x_6\}, s_2(x_1, x_3, x_4, x_6), I_1, ..., I_{10}, (\theta_{ij}))$. The local query at $\alpha_{\rm O}$ is $E{=}\pi_{x_1,x_3,x_4,x_6}$ $\sigma_F(s_1 \bowtie \pi_{\alpha_{\rm O}} Sys)$ where $F \equiv (x_2{=}x_9) \wedge (x_3{=}x_8) \wedge (100 \leqslant x_4) \wedge (16 \leqslant x_5) \wedge (x_6 \leqslant 1500)$. After E is evaluated, the result s_2 is sent to α_a since $\alpha_{\rm O}$ is the last source on the path t. ∎

Theorem 7.2 For each conjunctive query Q, the answer obtained by our evaluation methods is its definite answer.

8. Query Optimization

There are many interesting query optimization problems. Here we mainly consider global optimization techniques which allow us to reduce the size of query plans and the size of messages between data sources.

In general, optimization can be static or dynamic. Static optimization is done during plan generation; examples include finding the tightest interval for each essential variable, ordering subgoals based on binding patterns of variables and SDC optimization. The algorithms in §4 and §5 have included the first two kinds; SDC optimization will be introduced in §8.2. Dynamic optimization is done during query traveling. This is interesting because after some partial evaluation at a source, the result may suggest alternative ways for the subsequent evaluation. For example, if the result becomes empty, continued evaluation becomes unnecessary (Step 3 of Algorithm 3); otherwise, the interval sets may be further tightened and the remaining subgoals can be reordered according to the result.

Example 8.1 Consider an envelope (qid, α_a, s, t) where $t = (\alpha_1, r_1(x_1, x_2), \{x_1\}, s_1(x_2),\ ...,\ \alpha_2, r_2(x_3, x_4), \{x_3\}, s_2(x_4), ..., I_1{=}\{(0, 5]\}, I_2{=}I_3{=}\{(-\infty, 200)\}, I_4 = \{(-\infty, +\infty)\}, ..., (\theta_{ij}))$ and θ_{23} is "=". Suppose the evaluation of the local query at α_1 only produces tuples satisfying $x_2{<}100$. Clearly we can update both I_2, I_3 to $(-\infty, 100)$ since $x_2{=}x_3$. ∎

The dynamic optimization technique illustrated in the above example is called *constraint narrowing*. Basically, new constraints need to be extracted from the new partial result and are used to modify (narrow) those constraints in the current path envelope. Then the remaining subgoals are reordered based on the new binding information and the schemas of temporary relations are re-generated before relaying the envelope to the next data source. To extend the basic algorithm with constraint narrowing, we introduce two operations:

- *extract(E)* which extracts new constraints from a set of tuples E (a result to a local query) in terms of equality, minimum, maximum constraints.
- *narrow(t)* which modifies a path t based on the output of *extract(E)*. It is similar to the algorithms in §4 and §5. Specifically, it does the satisfiability checking, interval tightening and subgoal reordering.

8.1. Sharing of Evaluation Paths

Suppose $t_\ell{=}(\alpha_1^\ell, r_1^\ell, b_1^\ell, s_1^\ell, ..., \alpha_k^\ell, r_k^\ell, b_k^\ell, s_k^\ell, I_1^\ell, ..., I_n^\ell, (\theta_{ij})^\ell)$, $\ell{=}1, 2$, are two evaluation paths of a query Q. For each positive integer i, t_1 and t_2 are ***i-prefix sharable*** if (1) $\alpha_j^1{=}\alpha_j^2, r_j^1{=}r_j^2, s_j^1{=}s_j^2$ for $1 \leqslant j \leqslant i-1$, (2) $\alpha_i^1{=}\alpha_i^2, r_i^1{=}r_i^2$, and (3) E_j^1 and E_j^2 are equivalent for $1 \leqslant j \leqslant i-1$, where $E_j^\ell, \ell{=}1, 2$, is the constructed local query at the source α_j in Algorithm 3.

It is straightforward to extend i-prefix sharable to more than two paths. Also if a set of paths are i-prefix sharable, they are also j-prefix sharable for each $j \leqslant i$.

Proposition 8.2 For two given evaluation paths t_1, t_2 of a query and a positive integer i, it can be determined if t_1, t_2 are i-prefix sharable in polynomial time.

We briefly discuss how to perform partial evaluations to include prefix sharing. After constructing all paths of a query, the plan generator finds the prefix sharable relationships among those paths and organizes them into a "prefix" tree. The root of the tree is always the plan generator and the other nodes are data sources. Each path from the root to a leaf represents an evaluation path. If two evaluation paths share one (tree) path starting from the root with length i, they are i-prefix sharable. To construct the prefix tree, the set of paths are first partitioned based on 1-prefix sharing. Each partition has one common edge starting from the root. The child nodes are the first data sources in those common prefix. The label of the edge is the set of paths in the corresponding partition. Then each 1-prefix sharing partition is further partitioned based on 2-prefix sharing and the child nodes of 1-prefix sharing partition nodes are created for the second data sources. The labels for edges are assigned in the same way as before. This process will continue until k-prefix sharing where k is the length of each path (i.e. the number of predicates in the user query).

After constructing the prefix tree, the notion of an envelope can be extended to represent a set of paths with sharing and the partial evaluation algorithm can be altered to support the "splitting" of paths.

8.2. SDC Optimization

Optimization of the SDCs is also instrumental in reducing the overall communication cost for evaluating a user query.

To reduce the overall communication cost, we can optimize the SDCs for evaluation plan generation. The objective is to transform the SDC for each relation so that different constraint tuples describe disjoint sets of tuples in the corresponding relation. This can be achieved by Algorithm 4.

Algorithm 4: SDC Optimization

Step 1: For each attribute A and each site α, we build the following set of base intervals so that different base intervals are disjoint: $\{[a_i, a_i] \mid -\infty < a_i < +\infty, 1 \leqslant i \leqslant n\} \cup \{(a_i, a_{i+1}) \mid 1 \leqslant i < n\}$, where $a_1, ..., a_n$ is the result of sorting all the end points of all the intervals for A and α.

Step 2: For each site α, each d-tuple t in the SDC for α, and each attribute A, we find all the nonempty intersections of the interval $t.A$ with the base intervals for this attribute and this site. The cross product of the intervals for the different attributes gives the set of constraint tuples equivalent to t.

The running time of the SDC optimization algorithm is a polynomial in the length of the SDC.

9. Future Work

There are many interesting problems related to query optimization that remain to be studied. For example, how to minimize the number of paths and to reduce duplicates among different paths. In particular, the current SDC model does not have enough information for further optimization. It will be interesting to extend the model towards query optimization. Related to the schema integration aspect, SDC can also be extended to include data quality information (such as completeness [23, 19]), binding patterns [24, 16], and integrity constraints [5], which may also be used for optimization purposes.

References

[1] S. Abiteboul, H. Garcia-Molina, Y. Papakonstantinou, and R. Yerneni. Fusion query optimization. Technical report, Stanford University, 1996.

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

[3] Y. Arens, C. Knoblock, and W. Shen. Query reformulation for dynamic information integration. *Journal of Intelligent Information Systems*, 1996.

[4] Digital Library Initiative. *IEEE Computer*, May 1996.

[5] O. M. Duschka and A. Y. Levy. Recursive plans for information gathering. In *Proc. IJCAI*, 1997.

[6] Electronic Commerce and the Internet. *CACM*, June 1996.

[7] Electronic Commerce. *IEEE Computer*, May 1997.

[8] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, V. Vassalos, and J. Widom. The TSIMMIS approach to mediation: Data models and languages. *Journal of Intelligent Information Systems*, 1997.

[9] M.R.Genesereth, A.M.Keller, and O.M.Duschka. Infomaster: An information integration system. In *Proc. ACM SIGMOD*, 1997.

[10] S. Grumbach and J. Su. Dense order constraint databases. In *Proc. ACM Symp. on Principles of Database Systems*, 1995.

[11] S.Guo, W.Sun, and M.A.Weiss. Solving satisfiability and implication problems in database systems. *ACM TODS*, 21(2), 1996.

[12] J. Hammer, H. Garcia-Molina, J. Widom, W. Labio, and Y. Zhuge. The stanford data warehousing project. *IEEE Data Engineering Bulletin*, 6, 1995.

[13] R. Hull. Managing semantic heterogeneity in databases: A theoretical perspective. In *Proc. ACM Symp. on Principles of Database Systems*, 1997.

[14] R. Hull and G. Zhou. A framework for supporting data integration using the materialized and virtual approaches. In *Proc. ACM SIGMOD*, 1996.

[15] P. Kanellakis, G. Kuper, and P. Revesz. Constraint query languages. *Journal of Computer and System Sciences*, 51(1), 1995.

[16] C. T. Kwok and D. S. Weld. Planning to gather information. In *Proc. AAAI*, 1996.

[17] A. Levy, A. Rajaraman, and J. J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proc. VLDB*, 1996.

[18] A. Levy, D. Srivastava, and T. Kirk. Data model and query evaluation in global information systems. *Journal of Intelligent Information Systems*, 1995.

[19] A. Y. Levy. Obtaining complete answers from incomplete databases. In *Proc. VLDB*, 1996.

[20] A. Y. Levy, A. O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. In *Proc. ACM Symp. on Principles of Database Systems*, 1995.

[21] A. Y. Levy, A. Rajaraman, and J. J. Ordille. Query answering algorithms for information agents. In *Proc. National Conf. on Artificial Intelligence*, 1996.

[22] W. Litwin, L. Mark, and N. Roussopoulos. Interoperability of multiple autonomous databases. *ACM Computing Surveys*, 22(3):267–293, 1990.

[23] A. Motro. Integrity = validity + completeness. *ACM Transactions on Database Systems*, 14(4):480–502, 1989.

[24] A. Rajaraman, Y. Sagiv, and J. D. Ullman. Answering queries using templates with binding patterns. In *Proc. ACM Symp. on Principles of Database Systems*, 1995.

[25] M. T. Roth and P. Schwarz. Don't scrap it, wrap it! A wrapper architecture for legacy data sources. In *Proc. VLDB*, 1997.

[26] 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.

[27] M. Stonebraker, P. M. Aoki, W. Litwin, A. Pfeffer, A. Sah, J. Sidell, C. Staelin, and A. Yu. Mariposa: a wide-area distributed database system. *VLDB Journal*, 5(1):48-63, 1996.

[28] V. S. Subrahmanian, S. Adali, A. Brink, R. Emery, J. J. Lu, A. Rajput, T. J. Rogers, R. Ross, and C. Ward. Hermes: Heterogeneous reasoning and mediator system. Technical report, University of Maryland.

[29] J. D. Ullman. Information integration using logical views. In *Proc. Int. Conf. on Database Theory*, 1997.

[30] G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25(3):38–49, 1992.

[31] H. Zhu, J. Su, and O. H. Ibarra. Efficient evaluation of linear constraint queries with interval B^+-trees. *This proceedings*.