BEST EFFORT QUERY ANSWERING FOR MEDIATORS WITH UNION VIEWS

A Thesis by
Rowshon Jahan Papri

Bachelor of Science, Military Institute of Science and Technology, 2007

Submitted to the Department of Electrical Engineering and Computer Science
and the faculty of the Graduate School of
Wichita State University
in partial fulfillment of
the requirements for the degree of
Master of Science

July, 2011
BEST EFFORT QUERY ANSWERING FOR MEDIATORS WITH UNION VIEWS

The following faculty members have examined the final copy of this thesis for form and content, and recommend that it be accepted in partial fulfillment of the requirement for the degree of Master of Science with a major in Computer Science.

______________________________
Prakash Ramanan, Committee Chair

______________________________
Abu Asaduzzaman, Committee Member

______________________________
Achita Muthitacharoen, Committee Member
DEDICATION

To my sister-in-law, Dr Suraiya Simi Rahman. She is my inspiration to pursue higher education.
Acquire knowledge even if you have to travel far away from your home.
ACKNOWLEDGMENTS

I would like to thank my adviser, Dr. Prakash Ramanan, for his many years of thoughtful, patient guidance and support. Thanks are also due to Dr. Suraiya Simi Rahman and Dewan Roushon Jamir. Together their friendship and selfless role modeling have contributed to my professional development. I would also like to extend my gratitude to members of my committee, Dr. Abu Asaduzzaman and Dr. Achita Muthitacharoen, for their helpful comments and suggestions on all stages of this research.
Consider an SQL query that involves joins of several relations, optionally followed by selections and/or projections. It can be represented by a conjunctive datalog query $Q$ without negation or arithmetic subgoals. We consider the problem of answering such a query $Q$ using a mediator $M$. For each relation $R$ that corresponds to a subgoal in $Q$, $M$ contains several sources; each source for $R$ provides some of the tuples in $R$. The capability of each source are described in terms of templates. It might not be possible to get all the tuples in the result, $Result(Q)$, using $M$, due to restrictions imposed by the templates. We consider best-effort query answering: Find as many tuples in $Result(Q)$ as possible. We present an algorithm to determine if $Q$ can be so answered using $M$.

**Keywords:** Union of Views, Information Integration, Data Integration, Mediator, Capability based query answering
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Data Integration</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Types of Data Integration</td>
<td>1</td>
</tr>
<tr>
<td>1.2.1 Federated Database Systems</td>
<td>2</td>
</tr>
<tr>
<td>1.2.2 Data Warehouse</td>
<td>2</td>
</tr>
<tr>
<td>1.2.3 Mediator</td>
<td>3</td>
</tr>
<tr>
<td>1.3 Challenges in Information Integration</td>
<td>3</td>
</tr>
<tr>
<td>1.4 Mediation Architecture</td>
<td>4</td>
</tr>
<tr>
<td>1.5 Paper Outline</td>
<td>6</td>
</tr>
<tr>
<td>2. LITERATURE REVIEW</td>
<td>7</td>
</tr>
<tr>
<td>2.1 Query Plan Selection at Mediator</td>
<td>7</td>
</tr>
<tr>
<td>2.2 Capability Based Query Plan Selection</td>
<td>7</td>
</tr>
<tr>
<td>2.3 Related Work</td>
<td>8</td>
</tr>
<tr>
<td>3. DEFINITIONS AND NOTATIONS</td>
<td>10</td>
</tr>
<tr>
<td>4. ADORNMENT GRAPH</td>
<td>13</td>
</tr>
<tr>
<td>4.1 Definition</td>
<td>13</td>
</tr>
<tr>
<td>4.2 Adornment Graph for Single Source</td>
<td>14</td>
</tr>
<tr>
<td>4.3 Adornment Graph for Multiple Source with Union Views</td>
<td>14</td>
</tr>
<tr>
<td>5. RELATIONS WITH SINGLE SOURCES</td>
<td>16</td>
</tr>
<tr>
<td>5.1 Algorithm for Single Source</td>
<td>16</td>
</tr>
</tbody>
</table>
TABLE OF CONTENTS (Cont.)

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2 Lemma for Single Source Algorithm</td>
<td>17</td>
</tr>
<tr>
<td>5.3 Theorem for Single Source Algorithm</td>
<td>18</td>
</tr>
<tr>
<td>5.3.1 Theorem</td>
<td>18</td>
</tr>
<tr>
<td>5.3.2 Example</td>
<td>19</td>
</tr>
<tr>
<td>6. RELATIONS WITH MULTIPLE SOURCES</td>
<td>21</td>
</tr>
<tr>
<td>6.1 Definition and Notation</td>
<td>21</td>
</tr>
<tr>
<td>6.1.1 Notation</td>
<td>21</td>
</tr>
<tr>
<td>6.1.2 Example of Best Effort Answering</td>
<td>22</td>
</tr>
<tr>
<td>6.1.3 Definition</td>
<td>23</td>
</tr>
<tr>
<td>6.2 Algorithm for Multiple Source</td>
<td>26</td>
</tr>
<tr>
<td>6.3 Theorem for Algorithm for Multiple Sources</td>
<td>29</td>
</tr>
<tr>
<td>6.3.1 Example of ‘Case 1’</td>
<td>30</td>
</tr>
<tr>
<td>6.3.2 Example of ‘Case 2’</td>
<td>33</td>
</tr>
<tr>
<td>6.3.3 Example of ‘Case 3’</td>
<td>37</td>
</tr>
<tr>
<td>7. FUTURE WORK</td>
<td>43</td>
</tr>
<tr>
<td>8. CONCLUSIONS</td>
<td>44</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>46</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure-1</td>
<td>Federated Database Systems</td>
<td>2</td>
</tr>
<tr>
<td>Figure-2</td>
<td>Data Warehouse</td>
<td>3</td>
</tr>
<tr>
<td>Figure-3</td>
<td>Mediator Architecture</td>
<td>5</td>
</tr>
<tr>
<td>Figure-4</td>
<td>A node for a source at the M</td>
<td>13</td>
</tr>
<tr>
<td>Figure-5</td>
<td>Adornment Graph with an edge between two nodes</td>
<td>13</td>
</tr>
<tr>
<td>Figure-6</td>
<td>Adornment Graph with a user query node</td>
<td>14</td>
</tr>
<tr>
<td>Figure-7</td>
<td>Adornment graph of multiple source with union views</td>
<td>15</td>
</tr>
<tr>
<td>Figure 8</td>
<td>Adornment Graph Algorithm for Single Source</td>
<td>17</td>
</tr>
<tr>
<td>Figure-9</td>
<td>Example Adornment graph algorithm for single source</td>
<td>19</td>
</tr>
<tr>
<td>Figure-11</td>
<td>Adornment Graph of union views</td>
<td>22</td>
</tr>
<tr>
<td>Figure 14</td>
<td>Algorithm for Multiple Sources</td>
<td>27</td>
</tr>
<tr>
<td>Figure 15</td>
<td>Functions setToF(x), setPToF(x) and setToP(x)</td>
<td>27</td>
</tr>
<tr>
<td>Figure 16</td>
<td>Functions resAndDel(n) and notResButDel(n)</td>
<td>28</td>
</tr>
<tr>
<td>Figure-17</td>
<td>Example of Adornment graph for the ‘Case-1’</td>
<td>31</td>
</tr>
<tr>
<td>Figure-20</td>
<td>Example of Adornment graph for the ‘Case-2’</td>
<td>34</td>
</tr>
<tr>
<td>Figure-26</td>
<td>Example of Adornment graph for the ‘Case-3’</td>
<td>38</td>
</tr>
</tbody>
</table>
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1: Contents of Source R&lt;sub&gt;2&lt;/sub&gt;, S&lt;sub&gt;2&lt;/sub&gt; and T&lt;sub&gt;2&lt;/sub&gt;</td>
<td>42</td>
</tr>
</tbody>
</table>
LIST OF SYMBOLS

\( \pi \) Projection

\( \alpha \) Template

\( \beta \) Template

\( M \) Mediator

\( Q \) Query

\( f \) Free

\( b \) Bound
CHAPTER 1

Introduction

1.1 Data Integration

Data integration is a process that integrates several data sources or databases and provides an integrated physical or virtual view to the user [8]. In other words, it is a process that makes several databases or other data sources work together as if it is a single database or data source [2]. A simple example is when two companies merge, they need to combine their two different databases and make them work as one database. Data integration has become significant in a variety of situations in both commercial and scientific research. It appears with increasing frequency as the volume and the need to share existing data explodes [8]. There are many types of information integration such as federated database system, data warehouse and mediation. Mediation is a process that provides a virtual view of integrated databases or data sources to the user. It was first introduced in [5]. Many research works have been done on mediators such as large queries optimization in mediator, capability based mediator, computing the capabilities of mediators or capability based query plan selection. Many algorithms have been designed to answer the mediator query such as chain algorithm, partition algorithm etc. It has become the focus of extensive theoretical work, and numerous open problems remain unsolved. Our research work is focused on query answering at mediator for union views. When a relation is split into two or more data sources such that each split data source has the same attributes and placed into two different physical places with different set of tuples, then the union of all split sources is called the union views of that split relation. We use best effort approach for query answering at the mediator for union views.
1.2 Types of Data Integration

As discussed above, there are three types of information integration such as federated database system, data warehouse, and mediation.

1.2.1 Federated Database System

Federated database system integrates data by connecting databases with each other. Figure-1 shows a federated database system. This system is better for integrating small number of databases. If the number of databases increases then the number of interconnection also increases; if there are \( n \) database in the systems and each database needs to talk to other \((n-1)\) databases then the system requires \( n(n-1) \) pieces of code to support queries.

![Figure 1: Federated Database Systems [2]](image)

1.2.2. Data Warehouse

Data warehousing system extracts data from data sources and store the extracted data physically in one global schema. The architecture is shown in the figure below. Data warehouse is updated periodically after 24 hours or whenever a modification is done at the source side. The problem with this system is that it sometimes cannot provide most recent updated data to the user as the data is 24 hours old.
Moreover, it is complicated to update or reconstruct the data warehouse if the system integrates large number of databases.

Figure 2: Data Warehouse Architecture [2]

1.2.3 Mediator

Mediator system is a virtual view of the global schema. It does not store any data physically. As it does not have its own data, it queries the sources and provides the most recent updated data to the user. We will discuss mediator in detail in section 1.4. Figure 3 shows the mediator architecture in section 1.4.

1.3 Challenges in Data Integration

Data sources or databases are developed independently. They are different from each other in many ways, i.e. SQL languages, relational database, non-relational databases, different data types that might not support other DBMS. While integrating these independent databases, some heterogeneity problems arise.

A simple example of information integration can be when two similar companies merge. Then they need to combine their databases to make the two companies system work as one. Suppose the
merged company needs to run a simple SQL query such as SELECT * FROM EMPLOYEE to see all the employees of those two companies. But the attributes of EMPLOYEE table of the two databases can be different types (For example EMPLOYEE_ID attribute can be integer in one database and the other can be string). Sometimes the value of one field can be null and other cannot be null. To overcome these heterogeneity problems such as communication, query language, schema, different data type, value, semantic etc., information integration has different approaches. Mediator is one of them. Mediator supports a virtual view, or collection of views, that integrates several sources through a global schema which is a collection of common attributes of all the sources and other attributes.

If the data integration method is mediation then another problem arises: To determine whether the user query can be answered by the mediator and if so, how to answer the query efficiently and confirm that the mediator extracts the maximum answer for the query. An algorithm can determine whether the mediator can answer the user query or not and extracts all the possible tuples from all the data sources. Query plan selection and optimization at the mediator is still a challenging issue these days.

1.4 Mediation Architecture

Mediator supports a virtual view, or collection of views, that integrates several sources through a global schema which is a collection of common attributes of all the sources and other attributes [8]. The mediator does not store any data. So, it must get relevant data from its sources and use that data to answer the user’s query. Thus mediator sends a query to each of its wrappers, which in turn send queries to their corresponding sources. The mediator may send several queries to a wrapper or may not query all wrappers. Wrapper works as a translator for the sources. Wrapper takes the user query from the mediator and translates it to corresponding queries for the sources. The wrapper is created by a software named wrapper generator (the wrapper generator creates a table that holds the various query patterns contained in the templates, and the source queries that are associated with each). Wrapper queries the sources and sends back the query result to the mediator. Then mediator combines the results from all wrappers and
sends the combined result to the user. The following figure shows a mediator interacting with two sources and integrating those two databases.

![Mediator Architecture](image)

**Figure 3: Mediator Architecture [2]**

In the diagram above, mediator sends part of the user query to each wrapper (wrapper is a part of the mediator architecture) and the wrappers translate the user query into appropriate query for the corresponding source. For example, source\(_1\) in the above diagram is an oracle database and source\(_2\) is a MS SQL database. User query requires querying both the data sources and user query can be in any SQL language. Wrapper will translate the user query into query\(_1\), that is the sql+ for the oracle database and also will translate the user query into msSQL which is the SQL language for MS SQL database for source\(_2\). Wrapper will again translate the source result into the user query language.
1.5 Paper Outline

Our research work is based on the query answering algorithm that uses best effort approach for the capability based mediator with union views. We assume that the mediator is developed with wrapper that is independent of query languages which means the mediator can answer any available user query languages for the data source. We will focus on how to answer the user query if that query requires querying two separate sources as a union views. Our algorithm will support the mediator to make decide whether it can answer the query for the union views of a relation. Our work is a modified version of the chain algorithm [6]. We have extended its ability to handle union views for a relation.

In Section 2, we present literature review and related work. Section 3, we present our definitions and notations. In Section 4, we describe adornment graph to represent $M$ and $Q$. In Section 5, we present our algorithm for the case where each relation has only one source; this is essentially the chain algorithm of [6]. In Section 6, we present our algorithm for mediators with multiple sources. In section 7 we discuss future extensions of our research. Section 8 concludes.
CHAPTER 2
LITERATURE REVIEW

2.1 Query Plan Selection at Mediator

A mediator deals with huge user queries and many of them even cannot be answered due to certain restrictions. So it is essential to make sure that the user query which will be delivered by the mediator to a source can be answered by the source. Mediator has to have some sort of source capability information so that it can use that information to determine whether the source can answer the user query or not. This can be done by query planner. There are many types of query planner such as cost based or capability based query planner.

Query planning at the mediator level before actually processing the user query has to be done properly. One of the most efficient query planners at the mediator is the capability based query plan selection. Each source has the capability to answer certain queries. If mediator know the capability of the sources then mediator can determine whether the user query can be answered or not.

2.2 Capability Based Query Plan Selection

Capability based optimizer defines the capability of each source connected to the mediator. Each source has attributes with restrictions or conditions. Some sources might require a value for an attribute and does not require a value for another attribute or the value must not be specified for another attribute. User query might specify a value for an attribute which cannot be specified. In such case the mediator won’t be able to answer the query as a certain attribute must not be specified for a source but the user query has specified a value for that attribute. So it is convenient to determine whether the source can answer the user query before asking the source.
Each source has capability for each attribute which can be defined by the following notations.

**Notations [2]:**

1. $f$ (Free) means that the attribute can be specified or not, as we choose.
2. $b$ (Bound) means that we must specify a value for the attribute, any value is allowed

An algorithm is used to determine whether the user query can be answered or not by looking at source adornment which holds the capability of that source.

### 2.3 Related Work

Extensive research works have been done on mediator, but still many theoretical research opportunities exist in this area, such as how to join and optimize large queries in mediator or how to calculate the capability of a mediator or what is the best approach to answer the mediator query for the single source or a union views of the same relation. Chain algorithm, partition algorithm, etc. have been established to answer the mediator query.

Mediators were first introduced in [5]. Implementation of mediators was studied in [1]. Capability based query answering using mediators was studied in [4, 6]. In [6], the chain algorithm is presented to determine if a query $Q$ can be answered using a mediator $M$, when each relation has only one source. [2] (Chapter 21) presents a nice summary of various data integration methods, mediators, capability based mediation, and the chain algorithm of [6]. [7] Studied mediators with multiple sources for each relation. They presented an algorithm to characterize queries that can be answered using such mediators, when exact answer is desired. The problem we study here differs from theirs in that we want best-effort (not exact) query answering.

When mediator receives a query from the user then mediator has to decide how it is going to process the query to get all the possible tuples from each source that is related to user query. Mediator needs a query plan selection to do that. Capability based query plan selection is an approach that answers user query by asking individual source step by step and resolves one source depending on the outcome of a previously resolved source. Capability based query plan selection algorithm can have following steps.

1. First determine which sources can be resolved by looking at the templates.
2. If one or more source can be resolved then it will pick a random source and resolves that source.

3. After resolving the source, it will have some bindings for some attributes which may make some more sources resolvable.

4. Repeat this process until it reaches one of the following situations:
   a. The mediator has asked enough queries to the source to resolve all the conditions of the user query. This is called feasible plan.
   b. The mediator cannot construct anymore valid form of source queries which indicates that user query cannot be answered. It is impossible to answer the mediator query.

Previously, Chain algorithm was introduced which is a capability query plan selection algorithm to resolve the user query by dividing it into subgoals for each source. The chain algorithm is a greedy approach to resolve the user query. This algorithm answers the user query only for relation that has single source. A relation with single source refers to a source which does not have any other split or partial physical data source somewhere else. Section 5 describes more about chain algorithm for a relation with single source. We are describing chain algorithm using adornment graph in that section. But this algorithm does not work for relation with multiple sources at the mediator.
CHAPTER 3
DEFINITIONS AND NOTATIONS

We let $Q$ denote a conjunctive datalog query. An example of such a query is:

$$Q(x, w) \leftarrow R(x, y, z) \land S(z, v) \land T(y, v, w).$$

Variables $x$, $y$, $z$, $v$, and $w$ denote attributes; subgoals $R$, $S$, and $T$ denote input relations. We let the variable names stand for the corresponding attribute names; for example, we take $R$ to be a relation over the attributes $x$, $y$, and $z$. We let $\text{Result}(Q)$ denote the set of tuples in the result of this query, for a given input relations $R$, $S$ and $T$; this is the set of tuples in the head $Q(x, w)$. We allow for the possibility that $Q$ restricts some attributes to fixed sets of values. For example, in the query above, attribute $x$ might be restricted to be in the set $V_x = \{2, 7\}$, and attribute $v$ might be restricted to $V_v = \{4, 7, 9\}$; such attributes are called value-restricted attributes.

**Definition 3.1. [Useful value]** Consider an attribute $a$ appearing on the right hand side of a mediator query $Q$. A particular (constant) value for $a$ is *useful* (for a given set of relations) if that value can be used on the right hand side (in place of $a$), to obtain a result tuple for $Q$.

**Definition 3.2. [Useful tuple]** Consider a subgoal $R$ in a mediator query $Q$. A particular tuple in the relation $R$ is *useful* (for a given set of relations) if that tuple can be used on the right hand side (in place of $R$), to obtain a result tuple for $Q$.

Consider a relation $R(a_1, a_2, \ldots, a_k)$ with $k$ attributes. Such a relation would typically correspond to a subgoal in the body of a query $Q$. We consider a mediator system in which each relation has one or more sources.

**Definition 3.3. [Source]** A source $R_j$ for a relation $R$ contains some of the tuples in $R$.

Let $R_1, R_2, \ldots, R_m$ be the different sources for $R$, in a mediator system. We let the source name $R_j$ $(1 \leq j \leq m)$ also denote the set of all tuples at the source $R_j$. We take $R$ to be $\bigcup_{j=1}^{m} R_j$; the $R_j$ need not be disjoint.
If \( R_j \) is a source for a relation \( R(a_1, a_2, \ldots, a_k) \), we also denote \( R_j \) by \( R_j(a_1, a_2, \ldots, a_k) \).

**Definition 3.4.** [Adornment] An *adornment* for an attribute \( a_i \), in a source \( R_j(a_1, a_2, \ldots, a_k) \), is either \( b \) or \( f \); \( b \) stands for *bound*, and \( f \) stands for *free*.

**Definition 3.5.** [Template] A *template* for a source \( R_j(a_1, a_2, \ldots, a_k) \) is a sequence \( \alpha = x_1x_2\ldots x_k \) of \( k \) adornments. For \( 1 \leq i \leq k \), \( x_i \in \{b,f\} \) is the adornment for attribute \( a_i \) in \( \alpha \). If \( x_i = b \), we say that \( a_i \) is *bound* in \( \alpha \); else, \( a_i \) is *free* in \( \alpha \).

Each source \( R_j \) has a set of templates. Each template \( \alpha \) specifies a rule for obtaining some tuples from \( R_j \), as follows. For each bound attribute \( a_i \) in \( \alpha \), we must provide \( R_j \) a set \( V_i \) of one or more values for \( a_i \). Then, the source \( R_j \) would return all its *matching* tuples: Tuples \( t \in R_j \) such that \( a_i(t) \in V_i \), for all bound attributes \( a_i \).

**Example 3.1.** Consider a source \( R_1(x, y, z) \) with template \( bf b \). If we provide the source with a set \( V_x \) of allowed values for attribute \( x \), and a set \( V_z \) of allowed values for attribute \( z \), then the source would provide all the tuples \((a, b, c)\) that it has, such that \( a \in V_x \) and \( c \in V_z \).

**Definition 3.6.** [All-free template] A template \( \alpha \) for a source is called an *all-free template* if all attributes are free in \( \alpha \); i.e., \( \alpha = ff\ldots f \).

Note that an all-free template \( \alpha \) for a source \( R_j \) can be used to obtain all the tuples in \( R_j \).

**Definition 3.7.** [Mediator System] A *mediator system* \( M \) consists of several sources, for each of several relations. Each source has a set of templates.

**Definition 3.8.** \((M, Q)\) denotes a mediator system \( M \) and a query \( Q \). The subgoals in \( Q \) must correspond to (some of) the relations in \( M \).

**Definition 3.9.** [Resolving a source] Consider \((M, Q)\). Resolving a source \( R_j \) using template \( \alpha \) means obtaining all the useful tuples (and possibly some useless tuples) from \( R_j \), using \( \alpha \). This is achieved by providing \( R_j \) a set \( V_a \) that contains all the useful values (and possibly some useless values), for each bound attribute \( a \) in \( \alpha \). Resolving a source \( R_j \) means resolving \( R_j \) using any one of its templates.

**Definition 3.10.** [Resolving a relation] Consider \((M, Q)\). Resolving a relation \( R \) means resolving all the sources for \( R \).
**Definition 3.11.** [*Resolving an attribute*] Consider \((M, Q)\). *Resolving an attribute* \(a\) means obtaining a set of values for \(a\) that includes *all* the useful values for \(a\) (possibly along with some useless values). ●

Note that if we resolve a relation, then we can resolve all the attributes of that relation. For example, when we resolve \(R(x,y,z)\), we have obtained a relation \(R' (x, y, z) \subseteq R\) that *contains* all the useful tuples in \(R\). Then we can resolve \(x\) by obtaining the set \(V_x = \pi_x(R')\). Some tuples in \(R'\) (and so some values in \(V_x\)) might be useless, depending on the contents of the other relations.

The problem we consider in this paper is *best effort query answering*: Given \((M, Q)\), find as many tuples as possible, in the result \(\text{Result}(Q)\). We are not allowed to guess values for a bound attribute at a source; but we can use the values found (for an attribute) at one resolved source, to resolve another source.
CHAPTER 4

ADORMENT GRAPH

4.1 Definition

Given \((M, Q)\), an adornment graph is a graph \(AG(M, Q) = (N, E)\), where \(N\) is a set of nodes and \(E\) is a set of labeled edges. There is one node for each source in \(M\) that pertains to a subgoal in \(Q\); the templates for a source are said to belong to the corresponding node. In our figures, we show the templates for a source inside the corresponding node. Consider a source \(n_1\) at the mediator \(M\) with \(a, b\) and \(c\) attributes with the \(f, f\) and \(f\) adornments respectively. The source \(n_1\) is represented as node in the adornment graph as below. The top row in the node is adornment and the bottom row is the attribute names.

```
  f  f  f  
 /  a  b  c  
```

**Figure 4: A node for a source at the \(M\)**

There is an edge \(e\) from node \(n_1\) to node \(n_2\), labeled by an attribute \(a\), if there exists templates \(\alpha_1 \in n_1\) and \(\alpha_2 \in n_2\), such that attribute \(a\) is free in \(\alpha_1\) and is bound in \(\alpha_2\). This edge \(e\) signifies that if we get some tuples from the source \(n_1\), then we can feed all the values for attribute \(a\) found in these tuples, to source \(n_2\).

```
  n1  f  f  f  
     /  a  b  c  
   \   \   \   
     n2  b  f  
           /  a  d  
```

**Figure 5: Adornment Graph with an edge between two nodes**

In addition, there is a special node \(n_0\), (oval shaped) that corresponds to the mediator query \(Q\); it has an all-free template that corresponds to the value-restricted attributes in \(Q\). User defined values for the value-restricted attributes in \(Q\) is set in \(V_a\) where \(a\) is the value-restricted attribute. The \(n_0\) node can have
multiple value restricted attributes, i.e. $V_a$, $V_b$. \( n_0 \) has only outgoing edges. For each value-restricted attribute \( a \) there is an edge labeled \( a \) from \( n_0 \) to each node \( n \) that has a template in which \( a \) is bound. We let \( l(e) \) denote the label of edge \( e \).

### 4.2 Adornment Graph for Single Source

Consider a mediator \( M \) with three relations \( R(a, b, c) \) and \( T(a, d) \). Each relation has single source. Each source has one template; the source is denoted with its template as superscript; for example, source \( R \) with template \( f f b \) is denoted by \( R^{ffb} \). Let the mediator query has two value-restrict attributes \( a \) and \( c \). The query node will have set of values for attribute \( a \) and \( c \) in \( V_a \) and \( V_c \).

\[
R = R^{ffb}(a, b, c) \\
T = T^{bf}(a, d)
\]

![Adornment Graph with a user query node](image)

**Figure 6: Adornment Graph with a user query node**

### 4.3 Adornment Graph for Multiple Sources with Union Views

Consider a mediator \( M \) with three relations \( R(x, y, z) \), \( S(z, v) \) and \( T(y, v, w) \). Each relation has two sources. Each source has one template; the source is denoted with its template as superscript; for example, source \( R_2 \) with template \( f f b \) is denoted by \( R_2^{ffb} \).

Let
\[
R = R_1^{fff} \cup R_2^{ffb} \\
S = S_1^{ff} \cup S_2^{ffb} \\
T = T_1^{fff} \cup T_2^{bff}
\]

Let \( Q \) be the query
\[
Q(x, w) \leftarrow R(x, y, z) \land S(z, v) \land T(y, v, w), \text{ without any value-restricted attribute.}
\]
Figure 7 shows the adornment graph $AG(M, Q)$. Note that the node $R_1$ has a $fff$ template corresponding to the attributes $x, y$ and $z$, the node $S_1$ has a $ff$ template corresponding to the attributes $z$ and $v$ and the node $T_1$ has a $fff$ template corresponding to the attributes $y, v$ and $w$; the edges from $R_1$, $S_1$ and $T_1$ corresponding to these three attributes.

Any source (ex. $P_1(x, v, w)$) in $M$ that does not pertain to a subgoal in $Q$ will not be represented in $AG(M, Q)$.

**Definition 4.1.** [Resolving a node] Consider $AG(M, Q)$. Resolving a node $n$ using template $\alpha$ means resolving the corresponding source using $\alpha$. Resolving a node $n$ means resolving the corresponding source (see Definition 3.9).
CHAPTER 5

RELATIONS WITH SINGLE SOURCES

In this section, we consider \((M, Q)\), where each relation in \(M\) has only one source. To answer \(Q\), we try to resolve the sources corresponding to the subgoals in \(Q\), one by one, in some order. We use the set of useful values found for an attribute \(a\), at one resolved source, to resolve another source using a template in which \(a\) is bound. If any source cannot be resolved, then we say that \(Q\) cannot be answered; else \(Q\) can be answered. For the case under consideration (i.e., one source for each subgoal/relation), when \(Q\) can be answered, we get all the tuples in \(\text{Result}(Q)\); i.e., best effort query answering is the same as exact query answering.

5.1 Algorithm for Single Source

Our algorithm, Algorithm Single Source, given in Figure 8, determines if \(Q\) can be answered. The algorithm is similar to the well-known Chain Algorithm [6]. The main reason we are presenting it is that our algorithm for the general case, given in the next section, is an extension of it.

The algorithm changes the templates at each node in \(AG(M, Q)\), as it progresses. Consider any time instant during the execution of the algorithm. A node \(n_1\) with an all-free template can be resolved; the set of values obtained for an attribute \(a\) is then used to change \(a\) from bound to free, in the templates at as yet unresolved nodes \(n_2\). \(Q\) can be answered iff the algorithm can resolve all the nodes, one by one, in some order.

Algorithm Single Source

Input: \((M, Q)\), where each relation in \(M\) has a single source with one or more templates
Output: Whether \(Q\) can be answered using \(M\)
Construct \(AG(M, Q) = (N, E)\)
// Recall that a node \(n_0\) as having an all-free template corresponding to the value-restricted attributes in \(Q\) while \(\exists\) a node \(n_1\) with an all-free template do
{ If \(n_1 \neq n_0\) then resolve \(n_1\) using its original template \(a\) that now became all-free
// Propagate the attribute values from \( n_i \) to its adjacent nodes
for each edge \( e \) from \( n_i \) do
{
    let \( e = (n_1, n_2) \) and \( l(e) = a \)
    for each template \( \alpha_2 \) in \( n_2 \) for which \( a \) is bound, change \( a \) to free in \( \alpha_2 \)
}
delete node \( n_i \) and all its incident edges
}
if \( \exists \) a node \( n \) without an all-free template
then \( Q \) cannot be answered
else \( Q \) can be answered

**Figure 8: Adornment Graph Algorithm for Single Source**

### 5.2 Lemma for Single Source Algorithm

**Lemma 5.1.** Consider \((M, Q)\), where each relation in \( M \) has only one source. Consider any time instant during the execution of Algorithm Single Source. Suppose that a node \( n_i \neq n_0 \) has an all-free template. Then \( n_i \) can be resolved.

**Proof.** The proof is by induction on the time when \( n_i \) got an all free template. Let \( R \) be the source corresponding to \( n_i \), and let \( \alpha \) be the original template for \( R \) that became all-free at the time in question.

For the basis step, consider the first node \( n_i \neq n_0 \) that has an all-free template. There are two possibilities concerning \( R \).

**Case 1.** \( R \) is a source that started with an all-free template. Then, clearly, \( R \) can be resolved: We can get all the tuples in \( R \) from the source.

**Case 2.** \( R \) had an original template \( \alpha \) that had a bound adornment for some attributes; for each such attribute \( a \), there was an edge \( e_a = (n_0, n_i) \) labeled \( a \), that was used (by the algorithm) to change this adornment for \( a \) to free. Consider the construction of the adornment graph. The edge \( e_a \) implies that the query \( Q \) must have provided the set of allowed values for \( a \). For each such \( a \), this set of values for \( a \) can be provided to the source \( R \), to get all the matching tuples in \( R \); any other tuple in \( R \) is useless; so, \( R \) can be resolved.
Now, consider the induction step. By induction hypothesis, all nodes that got an all-free adornment before \( n_1 \) can be resolved; so, we can resolve each of the attributes in these nodes. Consider the original template \( \alpha \) at \( n_1 \) that became all free at the current time step. Consider an attribute \( a \) that had a bound adornment in \( \alpha \). Since \( a \) became all free, attribute \( a \) was resolved before the current time instant. So, we know all the useful values for each bound attribute in \( \alpha \). These values can be fed to the source \( R \) corresponding to \( n_1 \), to get all matching tuples from it; any other tuple in \( R \) is useless. So, \( R \) can be resolved.

Lemma 5.2. Consider \((M, Q)\), where each relation in \( M \) has only one source. Suppose that, during the execution of Algorithm Single Source, at the end of the while loop, there is a node \( n \) without an all-free template. Then \( n \) cannot be resolved.

Proof. Let \( R \) be the source corresponding to \( n \). For every template \( \alpha \) in \( n \), there must be an unresolved attribute \( a \). The nodes resolved up to this point do not contain this attribute; or else, our algorithm would have changed this attribute to free in \( \alpha \), using an edge \( e_a = (n_1, n) \), labeled \( a \), from a previously resolved node \( n_1 \). Since each unresolved node has such an attribute, none of them can be resolved.

5.3 Theorem for Single Source Algorithm

5.3.1 Theorem

Theorem 5.1. Consider \((M, Q)\) where each relation in \( M \) has only one source. There are two possible cases pertaining to the execution of Algorithm Single Source.

1. The algorithm resolves all the nodes, one by one, in some order. In this case, the query \( Q \) can be answered in a finite number of iterations, independent of the contents of the sources in \( M \).

2. At the end of the while loop, there is an unresolved node. In this case, \( Q \) cannot be answered.

Proof. Items 1) and 2) above follow from Lemmas 5.1 and 5.2, respectively. In item 1), the number of iterations (i.e., passes through the while loop) is the number of subgoals in \( Q \) plus one (the plus one is for \( n_0 \)), independent of the contents of the sources in \( M \).

18
5.3.2 Example

Consider the following example where relations $R(x, y, z)$ and $T(y, w, u)$ have single sources with the adornment templates $fff$ and $bff$ respectively.

Let mediator query be $Q(x,u) \leftarrow R(x,y,z) \land T(y,w,u)$ without any value-restricted attributes. The templates for the sources in mediator $M$ are:

$R = R^{fff}(x,y,z)$
$T = T^{bff}(y,w,u)$

Source $R$ can be resolved as its adornment template matches with the query subgoal template. It is noticeable that source $T$ adornment template does not match the mediator query subgoal adornment template. The subgoal $T^{fff}$ cannot be resolved by the source $T^{bff}$ as source adornment has $b$ adornment on attribute $y$ which cannot be resolved by the $f$ adornment on attribute $y$ of the query subgoal. But source $T$ can be resolved after resolving source $R$ which we will see later on in this example.

![Figure 9: Example Adornment graph algorithm for single source (example of above schema)](image)

**Figure 9: Example Adornment graph algorithm for single source (example of above schema)**

**Iteration 1:**

1. Source $R$ has an all free template. So it can be resolved. Set $V'_y = \pi_y(R)$
2. Fed $V'_y$ to the source $T$. As attribute $y$ is resolved so its adornment can be changed from $b$ to $f$.
3. Delete source $R$ and its outgoing edge from the adornment graph.

![Figure 10: Example Adornment graph algorithm for single source - step 1](image)
Iteration 2:

1. Source $T$ has an all free template. So it can be resolved.

2. As source $T$ do not have any incoming or outgoing edges, delete source $T$ from the adornment graph.

3. After step 2 no node exists in the graph. So the query is resolved.
CHAPTER 6
RELATIONS WITH MULTIPLE SOURCES

In this section, we consider mediators, where some relations have more than one source. We present an algorithm, Algorithm Multiple Sources, to determine if a given mediator query \( Q \) can be answered using the given sources.

Consider the case of relations with single source, studied in the previous section. When a source (i.e., relation) is resolved, we get all the useful values (possibly along with some useless values), for each attribute of that relation. In contrast, for a multi-source relation, when a single source is resolved, we only get a set of some (possibly none) useful values (possibly along with some useless values), for each attribute of that relation; whether any of these values is useful depends on the contents of other relations.

6.1 Definition and Notation

6.1.1 Notation

Up to now, an attribute can have one of two possible adornments: \( b \) and \( f \). Now, we introduce the third adornment \( p \); it stands for partially free. It is only used during the execution of the algorithm, and cannot be used to describe the original source capabilities.

We maintain a one-dimensional array \( adornStatus \) indexed by attributes \( a \). \( adornStatus[a] \) is independent of any template at any source, and can have one of three values:

- \( b \): No values for attribute \( a \) are known. In this case, we say that \( a \) is unresolved.
- \( f \): We have found a set of values for attribute \( a \); it consists of some (possibly none) useful values for \( a \), and some (possibly none) useless values for \( a \). Within our framework, we will not be able to find any more useful values for \( a \). In this case, we say that \( a \) is maximally resolved.
- \( p \): Similar to \( f \) above, except that we expect to find more useful values for \( a \). In this case,
we say that $a$ is partially resolved.

$\text{adornStatus}[a]$ is initialized to $b$ for all attributes $a$.

6.1.2 Example of Best Effort Answering

Consider a mediator $M$ with two relations $R(x, y)$ and $S(y, z)$. $R$ has two sources, each with a single template: $R_1^{bf}$ and $R_2^{bf}$; $S$ has one source $S_1^{bf}$.

Let $Q$ be the query $Q(x, z) \leftarrow R(x, y) \land S(y, z)$, with no value-restrict attributes. $AG(M, Q)$ is shown in Figure 11.

![Figure 11: Adornment Graph of Union Views](image)

Suppose that we first resolve $R_1^{ff}$ and get all its tuples. This gives us a set $V'_x = \pi_x(R_1)$ of values for attribute $x$, and a set $V'_y = \pi_y(R_1)$ for $y$. Since $x$ is bound in $R_2$ and does not appear in $S$, we cannot get any more useful values for $x$; so, we set $\text{adornStatus}[x]$ to $f$. But for attribute $y$, it is possible to get more useful values (see next paragraph), from $R_2$; so, we set $\text{adornStatus}[y]$ to $p$. After deleting $R_1$ from the graph along with its edges, we have:

![Figure 12: Adornment Graph of Union Views [step-1 for multiple sources algorithm]](image)
Now, we feed $V_x$ to $R_2^{bf}$, and get some more tuples in $R$. This maximally resolves $R_2$ and $R$. We project these tuples on $y$, and add the result to $V_y'$. This maximally resolves $y$ (since $y$ is bound in $S_1$); only now, we set $\text{adornStatus}[y]$ to $f$. Delete $R_2$ from the graph along with its edges as it is resolved.

$$\begin{array}{|c|c|}
\hline
S_1 & f \\
\hline
y & f \\
\hline
z & \\
\hline
\end{array}$$

**Figure 13: Adornment Graph of Union Views [step-2 for multiple sources algorithm]**

Finally, we feed all the values we have seen for $y$ to $S_1$, and maximally resolve $S_1$ and $z$. Delete $S_1$ from the graph as it has been resolved. The query is resolved as no node exists in the adornment graph.

Note that the result we get for $Q(x, z)$ is only the best-effort answer: We cannot get any more tuples, in our framework. It is not the full answer, however, because there could be other tuples in $R_2$ and $S_1$ (that we could not find) that might join to give more tuples in $\text{Result}(Q)$. This is because we maximally, but not fully, resolved $R_2$ and $S_1$. This situation does not arise in the previous section.

Extending Definitions 3.9–3.11, we have the following.

### 6.1.3 Definition

**Definition 6.1.** [Maximally resolving a source] Consider $(M, Q)$. Maximally resolving a source $R_j$ means obtaining a set $R_j' \subseteq R_j$ such that the following holds: It is not possible to obtain any more useful tuples from $R_j$ within our framework. This can be achieved by providing $R_j$ a set $V'_a$ obtained by maximally resolving attribute $a$, for each bound attribute $a$ in one particular template for $R_j$.

**Definition 6.2.** [Maximally resolving a relation] Consider $(M, Q)$. Maximally resolving a relation $R$ means maximally resolving all the sources for $R$.●
**Definition 6.3.** [Maximally resolving an attribute] Consider \((M, Q)\). Maximally resolving an attribute \(a\) means obtaining a set \(V'_a\) of values for \(a\), such that the following holds: It is not possible to obtain any more useful values for \(a\) within our framework.●

Note that if we maximally resolve a relation \(R\), then we can maximally resolve each attribute of that relation. For example, when we maximally resolve \(R(x, y, z)\), we have obtained a relation \(R'(x, y, z) \subseteq R\). Then we can maximally resolve \(x\) by obtaining the set \(V'_x = \pi_x(R')\). Some (or even all) tuples in \(R'\) (and so some, or even all, values in \(V'_x\)) might be useless, depending on the contents of the other relations.

Similar to the three definitions above, we have partial resolving

**Definition 6.4.** [Partially resolving a source] Consider \((M, Q)\). Partially resolving a source \(R_j\) means obtaining a set \(R'_j \subseteq R_j\); it is possible that \(R'_j\) is \(\emptyset\). This can be achieved by providing \(R_j\) a set \(V'_a\) obtained by partially resolving attribute \(a\), for each bound attribute \(a\) in one particular template for \(R_j\).

**Definition 6.5.** [Partially resolving a relation] Consider \((M, Q)\). Partially resolving a relation \(R\) means partially or maximally resolving at least one source for \(R\).

**Definition 6.6.** [Partially resolving an attribute] Consider \((M, Q)\). Partially resolving an attribute \(a\) means obtaining a set \(V'_a\) of values for \(a\); it is possible that \(V'_a\) is \(\emptyset\).

Note that if we partially resolve a relation \(R\), then we can partially resolve each attribute of that relation. For example, when we partially resolve \(R(x, y, z)\), we have obtained a relation \(R'(x, y, z) \subseteq R\). Then we can partially resolve \(x\) by obtaining the set \(V'_x = \pi_x(R')\). Some (or even all) tuples in \(R'\) (and so some, or even all, values in \(V'_x\)) might be useless, depending on the contents of the other relations.

Extending Definitions 3.6 and 4.1, we have the following.

**Definition 6.7.** [All-pfree template] template \(\alpha\) for a source is called an all-pfree template if each attribute is either free or partially-free in \(\alpha\).●
**Definition 6.8.** [Maximally/partially resolving a node] Maximally resolving a node means maximally resolving the corresponding source. Partially resolving a node means partially resolving the corresponding source.

**Definition 6.9.** [Supplier] A source $R_j$ is a supplier for an attribute $x$, if $x$ is free in some original template for $R_j$.

We let $adorn_{\alpha}(x)$ denote the adornment of attribute $x$ in a template $\alpha$ (at some node). $R(n)$ denotes the relation for which node $n$ is a source. *Algorithm Multiple Sources* is given in Figure 14. Like *Algorithm Single Source*, this algorithm changes the templates at each node, as it progresses. The algorithm calls several functions.

First, consider the functions $resAndDel(n)$ and $notResButDel(n)$ (see Figure 16). $resAndDel(n)$ maximally resolves a node $n$ that has an all-free template. It then passes attribute values from $n$ to adjacent nodes, and deletes $n$. $notResButDel(n)$ is called to delete a node $n$, when we cannot get any useful tuples from $n$; i.e., each template for $n$ has a bound attribute $y$, but $y$ has no maximally/partially resolvable supplier. The function updates the $adornStatus$ of some attributes from $p$ to $f$ (by calling $setPToF$), and then deletes $n$.

Next, consider the functions $setToF(x)$, $setPToF(x)$ and $setToP(x)$ (see Figure 15). $setToF(x)$ sets $adornStatus[x]$ to $f$, after attribute $x$ is maximally resolved. $setToP(x)$ sets $adornStatus[x]$ to $p$, after attribute $x$ is partially resolved. $setPToF(x)$ upgrades $adornStatus[x]$ from $p$ to $f$, after we find that we cannot get any more useful values for attribute $x$; it is called only from $notResButDel(n)$.

Consider any time instant during the execution of *Algorithm Multiple Sources*. A node with all free template can be *(maximally) resolved*: We can obtain all tuples from that source, using that template. Our algorithm repeatedly tries to partially/maximally resolve each node.

We prove the correctness of *Algorithm Multiple Sources* through a sequence of Lemmas.

**Lemma 6.1.** Consider the first while loop in the algorithm.

1. Any node $n$ that has an all-free template can be maximally resolved.
2. Any attribute $x$ that gets $adornStatus = f$ can be maximally resolved.
**Proof.** The proof is by induction, similar to the proof of Lemma 5.1. Not that, for an attribute $x$, we set

$$adornStatus[x] = f$$

only in two cases (see setToF($x$));

**Case 1.** $x$ is an attribute of a relation $R$ that has been *maximally* resolved; let $R' \subseteq R$ be the set of tuples we have obtained for $R$. Setting $V'_x = \pi_x(R')$ *maximally* resolves $x$.

**Case 2.** All suppliers of $x$ have been *maximally* resolved. Projecting the tuples we obtained for the suppliers of $x$ on $x$, *maximally* resolves $x$.

### 6.2 Algorithm for Multiple Sources

**Algorithm Multiple Sources**

**Input:** $(M; Q)$

**Output:** Whether $Q$ can be answered using $M$

Construct $AG(M; Q) = (N, E)$;

set $adornStatus[x] = b$, for all attributes $x$

// Recall that node $n_0$ has an all-free template corresponding to the value-restricted attributes in $Q$

while $\exists$ a node $n$ with an all-free template do

  // Maximally resolve $n$, and propagate the attribute values from $n$ to its adjacent nodes
  resAndDel($n$)

// Line A

for each node $n$ with an all-pfree template do

  // $n$ could be a node that got an all-pfree template during this loop

  // Partially resolve $n$, and propagate the attribute values from $n$ to its adjacent nodes

  for each attribute $x$ of $R(n)$ do

    setToP($x$)

// Line B

if $\exists$ any relation $R$ all of whose sources remain in the graph, and do not have an all-pfree template then

  // We cannot get any tuples from $R$ to answer $Q$

  $Q$ cannot be answered using $M$;

  exit

else $Q$ can be answered using $M$ // Continue with the algorithm

// Line C

for each node $n$ without an all-pfree template do

  // Cannot get any tuples from $n$; propagate this info to its adjacent nodes, and delete $n$

  notResButDel($n$)

// Line D

while $\exists$ a node $n$ with an all-free template do

  // Maximally resolve $n$ and propagate the attribute values from $n$ to its adjacent nodes

  resAndDel($n$)

// Line E

if all the nodes have been deleted

then $Q$ can be answered in a number of iterations that is independent of the contents of the sources.
else
{
    \exists \text{ a cycle in the (remaining) graph, consisting of nodes with all-pfree templates}
    Q \text{ can be answered in a number of iterations that depends of the contents of the sources}
}

**Figure 14: Algorithm for Multiple Sources [Union Views]**

`setToF(x)`

// Set adornStatus of attribute x to f
if adornStatus[x] \neq f
then
{
    adornStatus[x] = f
    for each relation R that has x as an attribute do
        for each source R_j for R do
            Set x to free in each template at R_j
            Delete all incoming edges at R_j with label x

}

`setPToF(x)`

// If adornStatus of attribute x is p, then change it to f
if adornStatus[x] = p
then
{
    adornStatus[x] = f
    for each relation R that has x as an attribute do
        for each source R_j for R do
            Set x to free in each template at R_j
            Delete all incoming edges at R_j with label x

}

`setToP(x)`

// If adornStatus of attribute x is b, then change it to p
if adornStatus[x] = b
{
    adornStatus[x] = p
    for each relation R that has x as an attribute do
        for each source R_j for R do
            for each template \( \alpha \) at R_j do
                if x is bound in \( \alpha \) then set x to partially-free in \( \alpha \)

}

**Figure 15: Functions `setToF(x)`, `setPToF(x)` and `setToP(x)`**
resAndDel(n)
// Maximally resolve n and propagate the attribute values from n to its adjacent nodes
if n is the last remaining source for R(n) in the graph
then
    for each attribute x of R(n) do:
        setToF(x)
else
{
    for each attribute x of R(n) do
        if n is the last remaining supplier for x
            then setToF(x)
        else setToP(x)
}
delete node n and all its incident edges

notResButDel(n)
// Cannot get any tuples from n; propagate this info to its adjacent nodes, and delete n
if n is the last remaining source for R(n) in the graph
then
    for each attribute x of R(n) do:
        setPToF(x)
else
{
    for each attribute x of R(n) do
        if n is the last remaining supplier for x
            then setPToF(x)
}
delete node n and all its incident edges

Figure 16: Functions resAndDel(n) and notResButDel(n)

Lemma 6.2. Consider the for loop between Line A and Line B. At any instant during this loop, a node n can be partially resolved iff it has an all-pfree template.

Proof. Throughout the execution of this loop, there is no node with an all-free template. Let α be a template at node n, at the instant under consideration; let x be an attribute that was originally bound in α. We would have set adornStatus[x] to p or f, at some prior instant, iff x was partially or maximally resolved, respectively.●

Lemma 6.3. Consider the time instant when the algorithm reaches Line B. Let x be an attribute with adornStatus[x] = b. x can never be partially or maximally resolved. Also, any node that has such a bound
attribute can never be partially or maximally resolved. So, the if statement following Line B correctly determines whether \( Q \) can be answered using \( M \).

**Proof.** Follows from Lemma 6.2.●

**Lemma 6.4.** The for loop between Line C and Line D deletes nodes that cannot be partially or maximally resolved. This might make some attributes that were previously partially resolved to become maximally resolved; the function call \( \text{notResButDel}(n) \) correctly makes the appropriate changes.

**Proof.** Follows from Lemma 6.3.●

**Lemma 6.5.** Consider the while loop between Line D and Line E.

1. Any node \( n \) that has an all-free template can be maximally resolved.
2. Any attribute \( x \) that gets \( \text{adornStatus} = f \) can be maximally resolved.

**Proof.** The proof is by induction, similar to the proof of Lemma 6.1.●

### 6.3 Theorem for Adornment Graph Algorithm for Multiple Sources

**Theorem 6.1.** Consider \((M, Q)\). There are three possible cases pertaining to the execution of Algorithm Multiple Sources.

1. The algorithm executes the then clause of the if statement following Line B. In this case, \( Q \) cannot be answered using \( M \); i.e., the best-efforts answer for \( Q \) is \( \emptyset \), irrespective of the contents of the sources.
2. The algorithm executes the then clause of the if statement following Line E. In this case, \( Q \) can be answered in a finite number of iterations, independent of the contents of the sources in \( M \).
3. The algorithm executes the else clause of the if statement following Line E. In this case, \( Q \) can be answered using \( M \), but the number of iterations depends on the contents of the sources in \( M \). In items 2) and 3) above: The best efforts answer might be \( \emptyset \), depending on the contents of the sources.
Proof. Item 1) follows from Lemma 6.3. Item 2) follows from Lemmas 6.1–6.5.; the number of iterations is the total number of passes through the various loops in the algorithm, independent of the contents of the sources in \( M \).

Consider item 3). Consider the graph \( G \) remaining at Line E. Each node \( n \) has an all-pfree template in which at least one attribute \( x \) is only partially resolved; i.e., \( \text{adornStatus}[x] = p \). \( n \) must have an incoming edge labeled \( x \), or we would have set \( \text{adornStatus}[x] \) to \( f \). So, \( G \) must have a cycle consisting of (some of) the remaining nodes (all of which have an all-pfree template). So, we partially resolve \( n \), feed values for some attribute \( y \) to the next node in the cycle, and so on. This process could continue around the cycle several times, until one source is exhausted.●

6.3.1 Example of ‘Case 1’

Consider a user query with no value restrict attributes. After executing the algorithm there can be a situation that the graph has a node with \( b \) adornment on an attribute and that attribute does not have any incoming edge from any other source. This indicates that source cannot be resolved by the user query. If the there no all-pfree node in the graph then it cannot be resolved.

Consider the following relation \( R \), \( S \) and \( T \) which was discussed in the above section. Each relation is the union of two separate sources. Notice that relation \( S \) has two sources \( S_1 \) and \( S_2 \) with \( u \) attribute whose adornment is \( b \) and has no incoming edge for that attribute.

\[
R = R_{1}^{fff} (x, y, z) \cup R_{2}^{ffb} (x, y, z) \\
S = S_{1}^{ffb} (z, v, u) \cup S_{2}^{pbb} (z, v, u) \\
T = T_{1}^{fff} (y, v, w) \cup T_{2}^{bff} (y, v, w)
\]

Let the mediator query \( Q \) with no value-restricted attribute,

\[
Q(x, w) \leftarrow R(x, y, z) \land S(z, v, u) \land T(y, v, w)
\]

The above query is asking for all the possible tuples from relation \( R \), \( S \) and \( T \). The mediator will make a decision before processing query at the source. The initial step is to construct the adornment graph for the \( M \). Only those sources which are in the query subgoal, will appear in the adornment graph.
In the above graph, relation $S$ has attribute $u$ whose adornment is $b$ and has no incoming edge for that attribute. This user query cannot be resolved.

**Iteration 1:**

1. There are two sources in $M$ with an *all-free* adornment. Let’s pick source $T_1$ (randomly).

   Resolve $T_1$ and set $V'_y = \pi_y(T_1)$ and $V'_v = \pi_v(T_1)$ with all the possible value of attribute $y$ and $v$ from $T_1$.

2. Now we have some binding of value from $T_1$ for attribute $y$ and $v$. Notices that source $S_2$ and $T_2$ has attribute $v$ and $y$ respectively with $b$ adornment. But both of the attributes has more incoming edges from another source. So they cannot be *maximally* resolved by $T_1$ but they can be *partially* resolved. So we change the adornment of attribute $v$ at source $S_2$ and the attribute $y$ at source $T_2$ from $b$ to $p$.

3. Delete the outgoing edges from $T_1$ along with the node $T_1$ itself from the adornment graph.
Figure 18: Example of Adornment graph for the ‘Case-I’ after step-1

Iteration 2:

1. There is one source $R_i$ in $M$ with an *all-free* adornment. Resolve $R_i$ and set $V'_y = (V'_y \cup \pi_y(R_i))$ and $V'_z = \pi_z(R_i)$.

2. Now we have some binding of value from $R_i$ for attribute $y$ and $z$. Notice that sources $R_2$ and $T_2$ have attributes $z$ and $y$ respectively, with $b$ and $p$ adornment respectively. But the source $R_2$ with attributes $z$ has more incoming edges from another source. So it cannot be *maximally* resolved by $R_i$ but it can be *partially* resolved. So we change the adornment of attribute $z$ at source $R_2$ from $b$ to $p$. Notice that the source $T_2$ with attribute $y$ has $p$ adornment and also has more incoming edge from another source. So we cannot change the adornment of attribute $y$ at $T_2$.

3. Delete the outgoing edges from $R_i$ along with the node $R_i$ itself from the adornment graph.
Figure 19: Example of Adornment graph for the ‘Case-1’ after step-2

After step 2 the above graph cannot be resolved as there is no all free adornment node and the other two partial resolved sources also cannot be resolved maximally as relation $S$ cannot be resolved for attribute $u$ as query or the other sources do not provide any value for that attribute.

6.3.2 Example of ‘Case 2’

Consider the following schema for mediator $M$ with relation $R$, $S$ and $T$; each is the union of two sources.

$$R = R_{1}^{fff} \cup R_{2}^{bfb}$$
$$S = S_{1}^{fff} \cup S_{2}^{fb}$$
$$T = T_{1}^{fff} \cup T_{2}^{bfb}$$

Let the mediator query $Q$ with no value-restricted attribute,

$$Q(x, w) \leftarrow R(x, y, z) \land S(z, v, u) \land T(y, v, w)$$

The above query is asking for all the possible tuples from relation $R$, $S$ and $T$. The mediator will make a decision before processing query at the source. The initial step is to construct the adornment graph for the $M$. Only those sources which are in the query subgoal, will appear in the adornment graph.
Iteration 1:

1. There are three sources in M with an \textit{all-free} adornment. Let’s pick source $R_1$ (randomly). Resolve $R_1$ and set $V'_y = \pi_y(R_1)$ and $V'_z = \pi_z(R_1)$ with all the possible values of attribute $y$ and $z$ from $R_1$.

2. Now we have some binding of value from $R_1$ for attribute $y$ and $z$. Notices that source $R_2$ has attribute $y$ and $z$ and $T_2$ has attribute $y$. But both of the attributes has more incoming edges from another source. So they cannot be \textit{maximally} resolved by $R_1$ but they can be \textit{partially} resolved. So we change the adornment of those two attributes from $b$ to $p$.

3. Delete the outgoing edges from $R_1$ along with the node $R_1$ itself from the adornment graph.

\begin{figure}
\centering
\includegraphics[width=\textwidth]{adornment_graph.png}
\caption{Adornment graph for the ‘Case-2’}
\end{figure}
Iteration 2:

1. There are two sources in $M$ are left with an all-free adornment. Let’s pick source $S_1$ randomly. Resolve $S_1$ and set $V_v = \pi_v(S_1)$ and $V_z = V_z \cup \pi_z(S_1)$

2. Now we have some binding of value from $S_1$ for attribute $v$ and $z$. Notices that source $R_2$ and $S_2$ has attribute $z$ and $v$ respectively with $p$ and $b$ adornment respectively. But both of the attributes has more incoming edges from another source. So they cannot be maximally resolved by $R_1$ but they can be partially resolved. So we change the adornment of attribute $v$ at source $S_2$ from $b$ to $p$. As attribute $z$ at source $R_2$ is $p$ so no need to change it.

3. Delete the outgoing edges from $S_1$ along with the node $S_1$ itself from the adornment graph.
Iteration 3:

1. There is only one source in $M$ is left with an all-free adornment. Resolve $T_i$ and set $V'_v = V'_v \cup \pi_v(T_i)$ and $V'_y = V'_y \cup \pi_y(T_i)$.
2. Now we have some binding of value from $T_i$ for attribute $v$ and $y$. Notice that source $R_2$, $T_2$ has attribute $y$ and source $S_2$ has attribute $v$ with $p$ adornment at the source side. Source $S_2$ has one more incoming edge on attributes $v$ so its adornment cannot be changed but source $R_2$ and $T_2$ don’t have any incoming edges from other sources on the attribute $y$, so we change its adornment from $p$ to $f$ as both of them are maximally resolved for attribute $y$. As attribute $v$ at source $S_2$ is $p$ so no need to change it.
3. Delete the outgoing edges from $T_i$ along with the node $T_i$ itself from the adornment graph.

![Adornment Graph](image)

**Figure 23:** Adornment graph for the ‘Case-2’ of Theorem after Step-3

Iteration 4:

1. There is only one source at $M$ has an all-free adornment. Resolve $T_2$ and set $V'_v = V'_v \cup \pi_v(T_2)$.
2. Now we have some binding of value from $T_2$ for attribute $v$. Notices that source $S_2$ has attribute $v$ with $p$ adornment. Source $S_2$ don’t have any incoming edges from any another sources on the attribute $v$ so change its adornment from $p$ to $f$ as it’s maximally resolved for attribute $v$.
3. Delete the outgoing edges from $T_2$ along with the node $T_2$ itself from the adornment graph.
Iteration 5:

1. There is only one source left in the adornment graph with an *all-free* adornment. Resolve $S_2$ and set $V'_z = V'_z \cup \pi_z(S_2)$.

2. Now we have some binding of value from $S_2$ for attribute $z$. Notices that source $R_2$ has attribute $z$ with $p$ adornment. Source $R_2$ do not have any incoming edges from any other sources on the attribute $z$ so change its adornment from $p$ to $f$ as it is *maximally* resolved for attribute $z$.

3. Delete the outgoing edges from $S_2$ along with the node $S_2$ itself from the adornment graph.

Iteration 6:

1. As the only left node in the graph so source $R_2$ with an *all-free* adornment. So resolve $R_2$.

2. As $R_2$ has all the free adornment with no outgoing edges so we can delete the node $R_2$ the adornment graph.

6.3.3 Example of ‘Case 3’

Consider a user query with no value restricted-attribute. The user query has an all-free template. After executing the algorithm, there can be a situation that the graph has a cycle. If the algorithm determines that there is a cycle in the graph then the query will be resolved after finite number of iteration.
but the number of iteration will not be equal to number of source node. The number of iteration depends on the content of the sources. In the worst case scenario the number of iteration is equal to the number of tuple in each relation. If a relation has total one millions of tuple them the query can be resolved after one millions of iteration.

For example let’s consider the following relation $R$, $S$ and $T$ at mediator $M$. Each relation is the union of two split sources.

$$R = R_1^{fff} \cup R_2^{bfb}$$
$$S = S_1^{fff} \cup S_2^{bfb}$$
$$T = T_1^{fff} \cup T_2^{bfb}$$

Let the mediator query $Q$ with no value-restricted attribute,

$$Q(x, w) \leftarrow R(x, y, z) \land S(z, v, u) \land T(y, v, w)$$

The above query is asking for all the possible tuples from relation $R$, $S$ and $T$. The mediator will make a decision before processing query at the source using the algorithm for multiple sources. The initial step is to construct the adornment graph for the $M$. Only those sources which are in the query subgoal, will appear in the adornment graph.

![Adornment graph](image.png)

**Figure 26: Example of Adornment graph for the ‘Case-3’**
Iteration 1:

1. There are three sources in \( M \) with an all-free adornment. Let’s pick source \( T_1 \) (randomly).
   Resolve \( T_1 \) and set \( V'_y = \pi_y(T_1) \) and \( V'_v = \pi_v(T_1) \) with all the possible value of attribute \( y \) and \( v \) from \( T_1 \).
2. Now we have some binding of value from \( T_1 \) for attribute \( y \) and \( v \). Notice that source \( T_2 \) and \( R_2 \) has attribute \( v \) and \( y \) respectively with \( b \) adornment. But both of the attributes has more incoming edges from another source. So they cannot be maximally resolved by \( T_1 \) but they can be partially resolved. So we change the adornment of attribute \( v \) at source \( T_2 \) and the attribute \( y \) at source \( R_2 \) from \( b \) to \( p \).
3. Delete the outgoing edges from \( T_1 \) along with the node \( T_1 \) itself from the adornment graph.

![Adornment Graph](image)

**Figure 27:** Example of Adornment graph for the ‘Case-3’ for Theorem after Step-1

Iteration 2:

1. There are two sources in \( M \) with an all-free adornment. Let’s pick source \( R_1 \) (randomly).
   Resolve \( R_1 \) and set \( V'_y = V'_y \cup \pi_y(R_1) \) and \( V'_z = \pi_z(R_1) \).
2. Now we have some binding of value from \( R_1 \) for attribute \( y \) and \( z \). Notice that source \( S_2 \) and \( R_2 \) has attribute \( z \) and \( y \) respectively with \( b \) and \( p \) adornment respectively. But the source \( S_2 \)
with attributes $z$ has more incoming edges from other source. So it cannot be maximally resolved by $R_i$ but it can be partially resolved. So we change the adornment of attribute $z$ at source $S_2$ from $b$ to $p$. Notice that the source $R_2$ with attribute $y$ has $p$ adornment and also has more incoming edge from another source. So we cannot change the adornment of attribute $y$ at $R_2$.

3. Delete the outgoing edges from $R_i$ along with the node $R_i$ itself from the adornment graph.

![Adornment Graph](image)

**Figure 28:** Example of Adornment graph for the ‘Case-3’ for Theorem after Step-2

**Iteration 3:**

1. There is only one source in $M$ left with an all-free adornment. Resolve $S_i$ and set $V'_v = V_v \cup \pi_v(S_i)$ and $V'_z = V_z \cup \pi_z(S_i)$.

2. Now we have some bindings for value from $S_i$ for attribute $z$ and $v$. Notice that source $S_2$ and $T_2$ has attribute $z$ and $v$ respectively with $p$ adornment. The source $S_2$ with attributes $z$ and the source $T_2$ with attributes $v$ have more incoming edges from other source. So it cannot be maximally resolved by $S_i$ but it can be partially resolved. But the adornment of the attributes of both of the sources is $p$ so we need not to change the adornment of the attributes.
3. Delete the outgoing edges from $S_i$ along with the node $S_i$ itself from the adornment graph.

![Adornment Graph](image)

**Figure 29: Example of Adornment graph for the ‘Case-3’ for Theorem after Step-3**

The graph above has a cycle. In the example above, after iteration 3, the three sources are *partially* resolved not *maximally* resolved. But the query can be resolved *maximally*. It will take much more iterations than *case 2*. The number of iteration can be less than or equal to the total number of tuples in the relation. In the *worst case* scenario, the number of iteration is equal to the total number of tuples in the relation.

Consider three sources $R_2$, $S_2$ and $T_2$ that has a cycle among them has the following values in the table.

<table>
<thead>
<tr>
<th></th>
<th>$R_2$</th>
<th>$S_2$</th>
<th>$T_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y</td>
<td>z</td>
<td>z</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
<th>z</th>
<th>v</th>
<th>w</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>8</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>9</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

**Table 1: Contents of Source $R_2$, $S_2$ and $T_2$**
The attribute which were partially resolved by the algorithm has value for attribute $y$ in $V_y = \{2\}$, for attribute $z$ in $V_z = \{3\}$ and for attribute $v$ in $V_v = \{1\}$.

**Step 1:** First if we query $R_2$ with $V'_y = \{2\}$ then we will get $V'_z = \{3\}$

**Step 2:** Then fed $V'_z = \{3\}$ value to the $S_2$ that will give us $V'_v = \{1\}$

**Step 3:** Then fed $V'_v = \{1\}$ value to the $T_2$ that will give us $V'_y = \{5\}$

**Step 4:** Then fed $V'_y = \{5\}$ value to the $R_2$ that will give us $V'_z = \{6\}$

**Step 5:** Then fed $V'_z = \{6\}$ value to the $S_2$ that will give us $V'_v = \{2\}$

**Step 6:** Then fed $V'_v = \{2\}$ value to the $T_2$ that will give us $V'_y = \{8\}$

**Step 7:** Then fed $V'_y = \{8\}$ value to the $R_2$ that will give us $V'_z = \{9\}$

**Step 8:** Then fed $V'_z = \{9\}$ value to the $S_2$ that will give us $V'_v = \{3\}$

**Step 9:** Then fed $V'_v = \{3\}$ value to the $T_2$ that will give us $V'_y = \{9\}$

The total number of iteration to complete the cycle is equal to the total number of the tuples of all sources.
CHAPTER 7

FUTURE WORK

In this work we have considered relations which are union of sources. Our algorithm picks a source with an all free template to resolve. If there are several sources that have all free templates, then we are picking the source randomly. But the order in which we resolve the sources could affect the run time of the algorithm. Future work would involve studying the best order to resolve the sources; this is cost based optimization.
In this paper, we considered the problem of answering a conjunctive datalog query $Q$ using a mediator $M$. [6] presented an efficient algorithm to determine if $Q$ can be answered using $M$, for the case where each relation in $M$ had only one source.

We considered the case where a relation could have many sources. In this case, it might not be possible to get all the tuples in the result, $Result(Q)$, using $M$, due to restrictions imposed by the templates. We considered best-effort query answering: Find as many tuples in $Result(Q)$ as possible. We presented an efficient algorithm to determine if $Q$ can be so answered using $M$. 
REFERENCES


