QUERY CONSOLIDATION: INTERPRETING QUERIES SENT TO INDEPENDENT HETEROGENOUS DATABASES

by

Aybar C. Acar

A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
In Partial fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy
Computer Science

Committee:

[illegible]	Dr. Amihai Motro, Dissertation Director
[illegible]	Dr. Alexander Brodsky, Committee Member
[illegible]	Dr. Carlotta Domeniconi, Committee Member
[illegible]	Dr. Larry Kerschberg, Committee Member
[illegible]	Dr. Daniel Menascé, Committee Member
[illegible]	Dr. Hassan Gomaa, Department Chair
[illegible]	Dr. Lloyd J. Griffiths, Dean, The Volgenau School of Information Technology and Engineering

Date: 7/23/08

Summer Semester 2008
George Mason University
Fairfax, VA

Query Consolidation: Interpreting Queries Sent to Independent Heterogenous Databases

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

By

Aybar C. Acar
Master of Science
Middle East Technical University, 2001
Bachelor of Science
Middle East Technical University, 1999

Director: Dr. Amihai Motro, Professor
Department of Computer Science

Summer Semester 2008
George Mason University
Fairfax, VA

Dedication

To my mother, who taught me to enjoy as I create.

And my father, from whom I am different, I hope, in no way that really matters...

Acknowledgments

I would like to start by thanking my advisor, Dr. Ami Motro, who has been a true friend and mentor from almost the first day I arrived at Mason. He has gone beyond the calling of an advisor, helping me with every aspect of my life at the department, as well as directing my research. Without his help and guidance this dissertation would not have been possible. I also extend my gratitude to the rest of the Motro family for the hospitality and graciousness they have shown me during my time here.

I would like to thank my committee members, Drs. Brodsky, Domeniconi, Kerschberg and Menascé, for their very valuable input and help. I am especially grateful to them and Dr. Hassan Gomaa, our department chair, for going out of their way to accomodate my defense process during the summer when it is so difficult to get everything organized. I also thank Dr. Sean Luke and the good people of the EC Lab for giving me a place to work and interact in the critical last months of my dissertation, when I suddenly found myself without an office. There are many other faculty and graduate students who have contributed their knowledge and ideas to my research over the years, I owe them all my gratitude.

I owe a great deal to all my family, starting with my parents, Feride and Ahmet Acar, who have comforted me when necessary, prodded me along when I needed it, provided direction when I was lost, and generally helped me the whole way with love. To my aunt and uncle, Emel and Ercan Acar, who have given me a home and hearth here in the States, been my first phone call in times of need, and have always provided me with loving support. Finally, my aunt and uncle back in Turkey, Fatoş and Murat Yazıcı, who have, with phone calls and visits, lightened my burden more than they know.

I would also like to thank all my friends and especially Kerem Çetinel, Gamze Ege and Kansu Dinçer, who have supported me, commiserated with me, helped me get perspective from time to time, and have generally done everything one could hope from life-long friends.

Last but definitely not the least, I owe a world of thanks to Ayşe Yazıcı, my friend, one-time roommate and sibling. She has been with me through it all; supporting me, suffering me, helping me, not to mention proofreading this dissertation, which must have read like gibberish, with tenacity befitting the great counselor I know she will become.

Table of Contents

List of Tables

List of Figures

Abstract

QUERY CONSOLIDATION: INTERPRETING QUERIES SENT TO INDEPENDENT HETEROGENOUS DATABASES

Aybar C. Acar, PhD

George Mason University, 2008

Dissertation Director: Dr. Amihai Motro

This dissertation introduces the problem of query consolidation, which seeks to interpret a set of disparate queries submitted to independent databases with a single "global" query. The problem has multiple applications, from improving virtual database design, to aiding users in information retrieval, to protecting against inference of sensitive data from a seemingly innocuous set of apparently unrelated queries. The problem exhibits attractive duality with the much-researched problem of *query decomposition*, which has been addressed intensively in the context of multidatabase environments: How to decompose a query submitted to a virtual database into a set of local queries that are evaluated in individual databases. The new problem is set in the architecture of a canonical multidatabase system, using it in the "reverse" direction. The reversal is built on the assumption of conjunctive queries and source descriptions. A rational and efficient query decomposition strategy is also assumed, and this decomposition is reversed to arrive at the original query by analyzing the decomposed components

The process incorporates several steps where a number of solutions must be considered, due to the fact that query decomposition is not injective.

Initially, the problem of finding the most likely join plan between component queries is investigated. This is accomplished by leveraging the referential constraints available in the underlying multidatabase, or by approximating these constraints from the data when not available. This approximation is done using the information theoretic concept of conditional entropy. Furthermore, the most likely join plans are enhanced by the expansion of their projections and adding precision to their selection constraints by estimating the selection constraints that would be applied to these consolidations offline.

Additionally, the extraction of a set of queries related to the same retrieval task from an ongoing sequence of incoming queries is investigated. A conditional random field model is trained to segment and label incoming query sequences. Finally, the candidate consolidations are re-encapsulated with a genetic programming approach to find simpler intentional descriptions that are extensionally equivalent to discover the original intent of the query.

The dissertation explains and discusses all of the above operations and validates the methods developed with experimentation on synthesized and real-world data. The results are highly encouraging and verify that the accuracy, time performance, and scalability of the methods would make it possible to exploit query consolidation in production environments.

Chapter 1: Introduction

Loosely coupled data integration among networked sources has become so ubiquitous over the recent years that many of the services and applications used daily are actually not monolithic information systems but rather collections of sources tied together. Instead of building centralized and large data sources (i.e., the Extract-Transform-Load method), many organizations and individuals are opting for a virtual database approach. Especially along with the advent of XML based service-oriented architectures, it has become very easy to leave data in its original source and to instead recruit the service provided by that source as needed. This structure is seen in a variety of scenarios such as hybrid web applications (mash-ups), enterprise information integration models, aggregation services and federated information retrieval systems. Furthermore, individual users are often forced to procure and assemble the information they need from sources distributed across a network.

Consequently, a good amount of work done to obtain the answers, including intermediate results, sub queries and the sources used, are observable to some degree to a third party. This presents a new opportunity for monitoring query activity from the outside. When an individual requires information or when a integration system is given a query, they are forced to disassemble their query into components and send them over the network to the relevant data sources and combine the answers into what they need. This party will for the rest of this dissertation be termed the *querying agent*. I propose that an outside party, the *monitor*, will be, to some extent, able to guess at the goal of the querying agent.

1.1 Motivational Examples

Assume one has a chance to observe the actions of a particular user. This user first connects to the website of the piano technicians guild and downloads a table listing the names and phone numbers of registered piano tuners. Next, the user goes to the phone company website and retrieves a list of area codes for the New York area (212, 917 &c.). Finally, the user searches for the names of piano tuners having good ratings from a customer review site for piano enthusiasts.

These transactions can be viewed as three separate relations:

1. Names and phones of registered tuners from the piano technicians guild.

2. Area codes in the New York area from the phone company.

3. Names of 'good' tuners from the review site.

A monitor observing these transactions will come to the conclusion that the user is trying to find a good piano tuner that operates in the New York area. This is the consolidation of these queries. Looking at it from the relational algebra point of view, if the component relations listed above are named R_1 to R_3, respectively, the consolidation would be:

$$R_1 \bowtie R_2 \bowtie R_3$$

Another example illustrates a slightly different scenario. Assume now that the querying agent is a person trying to organize her vacation from Washington D.C. to some resort in the Caribbean. She does not necessarily care which resort it is as long as it is in the Caribbean. Unfortunately the flight booking website requires distinct airports to be specified to search for flights. Therefore, she first retrieves a listing of airports from IATA with the constraint that the airports be in the Caribbean region. Then, one by one, searches for flights from Dulles International to each of these airports. Watching these queries, the monitor would decide that the proper consolidation is the list of flights from D.C. to Anguilla, or to Aruba, or to Bermuda and so forth. In relational algebra terms, if one names the list of airports R_1 and the result of the search for each airport R_2 to R_n the consolidation would be:

$$R_1 \bowtie (R_2 \cup R_3 \ldots \cup R_n)$$

A final example illustrates the recursive aspect of query plans that may be employed by querying agents. Consider a software program tasked with building a bibliography on some subject. Assume the citation server it connects to allows only searches by paper ID. Once an ID is given the server returns the list of paper IDs and information for the papers that the original paper cites. Since there is no way for the querying agent to ask for all the papers at once, it will need to extract the list, or as much of it as it can by traversing the citations. Starting from a known paper ID, the software agent retrieves the bibliography of that paper and then one by one retrieves the citations of the papers it has just received and so forth until it gets no new results (i.e., the recursion arrives at a fixpoint). The consolidated result is the list of all the papers it has seen up to the point that it can find no more. Again, if one terms the answers of each of its queries as R_1 to R_n, the consolidation is:

$$R_1 \cup R_2 \ldots \cup R_n$$

1.2 The Problem

The problem being addressed is: Given a set of queries which are the result of the decomposition of a global query across multiple independent databases, can we recreate the global query from these pieces? One can approach the problem by using the well-researched architecture of virtual databases.

Briefly, a virtual database architecture consists of a set of local databases $L_1, \ldots, L_n$, a global database scheme G, and a mapping of the global scheme into the local databases. The conventional way of using this architecture is as follows:

1. A query Q is submitted to the global database G;

2. Q is decomposed to a set of queries $Q_1, \ldots, Q_n$ on the local databases $L_1, \ldots, L_m$;

3. Each query Q_i is submitted to one of the databases L_j resulting in an answer A_i;

4. The answers $A_1, \ldots, A_n$ are assembled in an answer A to the original query Q.

This entire process may be summarized as a *query decomposition* problem:

$$\boxed{\text{Given a global query } Q\text{, find local queries } Q_1, \ldots, Q_n \text{ and an expression } E \text{ such that } Q \supseteqq E(Q_1, \ldots, Q_n).}$$

Notice that the global query Q is not necessarily always equal to the result of the decomposition. Depending on the available views, an arbitrary query may only be partially answerable with any expression. Therefore, the result of the expression E is equivalent or *maximally contained* in the answer of the global query Q. I use the '$\subseteqq$' and '$\supseteqq$' symbols throughout this document to denote maximal containment. The maximal containment concept and related issues will be further discussed in Chapter 3.

For the purpose of interpreting a set of local queries, I adopt the same architecture, but consider a task which is the *inverse* of the earlier task, and which is thus named the *query consolidation* problem:

$$\boxed{\text{Given local queries } Q_1, \ldots, Q_n\text{, find a global query } Q \text{ and a related expression } E \text{ such that } E(Q_1, \ldots, Q_n) \subseteqq Q.}$$

In the "forward" direction (i.e., query decomposition), there could be many alternative decompositions; i.e., one could have both $Q = E(Q_1, \ldots, Q_n)$ and $Q = E'(Q'_1, \ldots, Q'_n)$. In such cases a query optimizer will normally prefer the solution in which the local queries retrieve only data that are necessary to answer Q.

Example 1.1. Consider a virtual database $R = (B, C, D)$ and two local database relations $R_1 = (A, B, C)$ and $R_2 = (C, D, E)$, and let $Q = \pi_B \sigma_{D=3}(R)$.

Consider these two decompositions:

1.	2.
$Q_1 = R_1$	$Q_1 = \pi_{B,C}(R_1)$
$Q_2 = R_2$	$Q_2 = \pi_C \sigma_{D=3}(R_2)$
$E = \pi_B \sigma_{D=3}(Q_1 \bowtie Q_2)$	$E = \pi_B(Q_1 \bowtie Q_2)$

The first decomposition retrieves the entire relations R_1 and R_2, whereas the second decomposition retrieves only the information necessary to evaluate Q. In other words, the assembly process of the first decomposition discards much more data. Well-known techniques of query optimization may be used to improve a given solution [86]. □

Similarly, in the "backward" direction (i.e., query consolidation), one should also look for an expression E (and its implied query Q) that removes as little data as possible; i.e., a global query Q that will interpret as much of the data retrieved by $Q_1, \ldots, Q_n$ as possible.

In the previous example, the query $Q = \pi_B \sigma_{D=3}(Q_1 \bowtie Q_2)$ is a poor interpretation of the queries $Q_1 = R_1$ and $Q_2 = R_2$, as it removes much of the data retrieved by the two queries. On the other hand, the query $E = \pi_B(Q_1 \bowtie Q_2)$ is a good interpretation of the queries $Q_1 = \pi_{B,C}(R_1)$, $Q_2 = \pi_C \sigma_{D=3}(R_2)$, as it removes only a small amount of retrieved data.

These observations may be summarized as follows.

A query Q is a consolidation of a given set of local queries $Q_1, \ldots, Q_n$ if Q can be decomposed into $Q_1, \ldots, Q_n$ by optimal query decomposition.

1.3 Applications

The goal of the querying agent requires information that must be obtained from different sources, and the information is then assembled off-line. There are two principal motivations for such activities. One motivation is innocent. The information this user requires is dispersed over multiple databases, forcing the user into the laborious process of submitting individual queries to different databases and then assembling and correlating the information off-line. Discovering an interpretation for this query set may help suggest how information could be reorganized to facilitate such tasks in the future. Indeed, the main argument for constructing virtual databases has always been to provide in a single source all the information necessary for a particular task [69]. Discovering interpretations for distributed query sets may suggest useful reorganizations and consolidations, either physical or virtual. The applications can be grouped under several more specific aims:

Distributed Re-design. Consider a shopping mall with multiple stores, and assume that an analysis of sale records shows that within a small time interval, the same customer purchased a box of candy in one store, gift-wrapping paper in another, and a greeting card in a third. A global interpretation of this local information may suggest that service could be improved if these three items where to be sold in the same store. Similarly, query consolidation may suggest redesign of available information sources to correspond more efficiently to popular information needs.

Targeted Advertising Similar to the previous application, the individual requests of a user can give the system information as to the general goal. This in turn will allow an associated advertising mechanism to offer products or services that more relevant to the task the user wants to achieve than the individual parts. The example of the shopping mall is again pertinent. Instead of offering different gift-wrapping papers and candy selections as the user browses these, a more precise and thus enticing offer is possible if the individual parts are consolidated. If the system can predict the global task (i.e., getting a gift for someone), it can offer pre-packaged gift selections instead, which are more in tune with the users current situation.

User-Database Interface Improvement. The problem described could easily be adapted to a situation in which the input set of queries is obtained from the log of a single database; i.e., a set of queries submitted by a single user to a single database. The reason for such a set may be that the user-database interface does not permit arbitrary queries, and the user is forced to submit multiple queries, each usually small, and then assemble the information off-line. A good example to this is the case of the person trying to book a flight to the Caribbean, given earlier. A query consolidation analysis may discover the overall purpose of the query set, and possibly lead to a re-design of the query database interface to facilitate future inquiries.

User Education. A different, though not entirely unrelated, reason for query sets, in either the distributed or centralized cases, is user inexperience or ignorance. In the distributed case, the user might be submitting a set of queries to different databases and correlating them off-line, when the same goal could be achieved by accessing a single database. In the centralized case, the user might be submitting a set of small queries and assembling them off-line, when the same goal could be achieved with a single query, perhaps using a feature of which the user is not aware. A query consolidation analysis may suggest flaws in the way the system is advertised or in the training of its users. This application is reminiscent of other systems that track user behavior and suggest improvements, such as office software or online stores.

The second motivation for leads to the interesting application of query consolidation in surveillance and security. The consolidated queries imply the intentions and motives of the querying agent. The actual inference of these intentions from consolidated queries is a task for a human expert but a query consolidation system can do the tedious work of consolidation. Since there will probably be a large number of users each with a lengthy history of queries, the function of the query consolidator will be to sift through the logs and compile the simplified forms of likely consolidations and present them to the expert. This will allow a security expert to infer the intentions by looking at the distilled form without having to assemble and prune of a possibly huge number of queries manually. A variety of options are available: The expert can focus on a single user and essentially get a listing of interests during a time period. Alternatively, trends can be analyzed across many sources looking for a certain intention shared by a group of users. Query consolidation will also be useful as a detection mechanism if the possible intentions and the global queries that imply them are known in advance. The system can be set up so that certain information is on a watch-list and any consolidation of queries that significantly overlap that information can be flagged automatically by the system, along with the users that posed said queries (e.g., "Did anyone want to compile this information and if so, how did they go about it?"). This will undoubtedly have its uses in national security and certain commercial settings where confidentiality is at a premium. Again, there are a couple of more specific ways that this application can be elaborated:

Detecting Malevolent Intent. By consolidating a set of independent queries submitted to different databases,one may be able to detect illegal or suspicious requests that could otherwise go undetected. As a simple example, assume ($\underline{Person}$,$Rank$) and ($\underline{Rank}$,Pay) are retrieved from two different databases. A possible global interpretation may include the association ($Person$,Pay), a disclosure which might be undesirable. To discover such activities requires that a set of "protected" concepts be pre-declared, and that the discovered consolidations be compared to this set for possible infringements.

Global Inference Control. Inference control is a well-researched subject that concerns prevention of situations in which information obtained legally from a database is used, possibly (but not necessarily) in conjunction with "outside" knowledge, to infer protected information. As a simple example, the two facts "The patient Jones lives in Springfield" and "only two of the hospital's cancer patients live in Springfield" may be combined to infer that "With probablility 0.5 Jones has cancer". Attempts to detect inference are greatly handicapped in a distributed environment, where it is hard to keep tab of the information made available to attackers. By linking together independent databases by means of a virtual database, inference control mechanisms may be applied to consolidated information, thus providing greater protection.

1.4 Research Issues

The problem of query consolidation presents significant challenges. These can be classified into five main research issues that are handled to some extent in this dissertation.

Solution Multiplicity. In practice, the implemented query decomposition procedure may not be truly optimal, but in defining a deterministic procedure, query decomposition becomes a *function* (i.e., single-valued). Nonetheless, it is not necessarily an injective function (i.e., one-to-one). Hence, there is no unique inverse.

In the environment of Example 1.1, consider the global queries $Q = \sigma_{B>D}(R)$ and $Q' = \sigma_{B<D}(R)$. Both would require the same two component queries $Q_1 = \pi_{B,C}(R_1)$ and $Q_2 = \pi_{C,D}(R_2)$. Hence, given Q_1 and Q_2, both Q and Q' are possible consolidations.

Consequently, query consolidation requires additional techniques for determining the *most likely* consolidations for a given set of local queries. These techniques would involve the application of additional knowledge on likely search targets, as well as the development of ranking procedures.

Input Extraction. The statement of the problem assumes a given set of local queries $Q_1, \ldots, Q_n$. In practice, however, each of the local databases can be assumed to have a *log* of queries submitted to that database. A practical problem is to extract a set of queries from the n logs that corresponds to a single task submitted by a single querying agent.

There are two dimensions to the problem. The first issue is to ascertain the identity of the agent. There are multiple pieces of information associated with every transaction that allows one to identify individual parties. These include connection information such as IP addresses, authentication details such as usernames and passwords and session tagging mechanisms such as cookies. In the current computing and networking environment it is generally not very difficult to identify a certain individual unless the party concerned is going to the trouble of hiding their identity. Notice that, if such is the case most of the mechanisms mentioned can be circumvented by using proxies and periodically resetting session information. Although explicit user authentication is more difficult to circumvent it is still possible to confuse identification by several individuals colluding to share the workload of a suspicious query plan among themselves.

Given this situation, the issue of identifying individuals has been kept out of the scope of the research. Ultimately, this sort of identification is either trivial or extremely difficult. Furthermore, the difficulty stems from a matter of security policy and enforcement of authentication procedures and is largely out of the scope of this research. Therefore, I assume that there is a universal ID associated with each user and as such each query in each log is identifiable as belonging to one agent or another. Even when the origins of the queries are known, input extraction still remains a problem as there is the issue of segmenting the input coming from an individual agent into subsequences denoting individual tasks or "sessions". As an example consider the case of the person looking for a piano tuner mentioned in Sec. 1.1. During the lifetime of that user they will necessarily ask many queries. For example, the same person may inquire about current movies after finishing their task regarding the piano tuners. A query consolidation system should be able to distinguish the fact that one task is finished and a new one has started.

Input Imperfection. In practice, the set of local queries $Q_1, \ldots, Q_n$ may be incomplete; that is, the global query that is the true intent of the user encompasses additional queries, which may be unavailable to the system for a variety of reasons: The user may already have the answers to these queries, or he may have submitted them to databases that are "outside" the system, or the monitoring system missed them for some unknown reason.

Similarly, the input set $Q_1, \ldots, Q_n$ may be unsound, including queries and tuples that are not part of the eventual goal of the user. The reason for the inclusion of such unrelated queries could be innocent (e.g., user error or lack of granularity in the querying mechanism) or malevolent (e.g., to obscure the true intent of the user). Obviously, the presence of such "noise" is a hindrance to the discovery process.

An important research issue is how to assure that the discovery process is robust, and can withstand moderate levels of input imperfection (i.e., incompleteness and unsoundness).

Real-time Analysis. The initial definition of the treatment of the query consolidation problem (Sec. 1.2) assumed that the entire set of input queries is available before the analysis begins. In some of the applications, in particular those that concern security and privacy, it would be desirable to be able to conclude that a suspicious or malevolent attack is forming as soon as possible. Indeed, one could envision a tool that tracks query logs continuously, updating and refining its conclusions, and signaling an “alert” as soon as it concludes that a suspicious goal is being attempted. The required facilities for such operation are an evolving model and fast consolidation algorithms.

Precision vs. Conciseness. Two desirable properties of the global query Q are *precision* and *conciseness*. Precision means that Q corresponds to the local queries $Q_1, \ldots, Q_n$ precisely; i.e., the optimal decomposition of Q is precisely $Q_1, \ldots, Q_n$. Conciseness means that Q is brief and simple. There is a well known trade-off between these two properties [85]. That is, to achieve good precision may require a query that is complex and unnatural, whereas there may exist a much simpler interpretation that corresponds less perfectly to $Q_1, \ldots, Q_n$. Compact yet imperfect interpretations are attractive because the data retrieved by $Q_1, \ldots Q_n$ may include noise: small amounts of data that were retrieved even though they are not part of the overall goal, and vice-versa, small amounts of data in the overall goal that were not retrieved. In essence, having a very precise and bloated expression may be seen as overfitting and may lose the generality of the original goal.

1.5 Operational Scenarios of the Monitor

The system can presumably monitor queries posed to independent sources in one of two modes depending on what is being monitored.

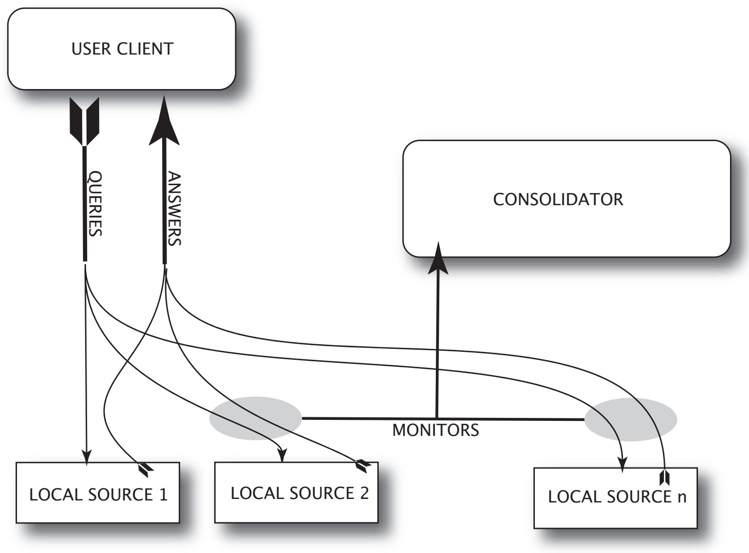


Figure 1.1: **Object Monitoring Mode**

Object Monitoring

In object monitoring (Figure 1.1) the system is set up to monitor the query logs of local data sources[1]. This is analogous to monitoring the records of a library for checkout records. The advantage is to be able to monitor the use of sensitive data located on the local sources by anybody, as opposed to monitoring users one by one. Therefore, the main application of this mode security and privacy. There are two primary difficulties associated with this mode including:

When looking at queries coming into a source, one needs to separate the interleaved queries into separate threads which isolate different users. If an authentication mechanism is used by the source, this authentication can be used for threading. However, if the source is unauthenticated or there is a conspiracy of users sharing various sub-queries of a concept, user identification with authentication information is not possible. Even when the user is unequivocally identified, there is the problem of session or concept identification.

When the sources themselves are monitored there is always the chance that the user

[1]Note that in the example in Figure 1.1 the monitor is not able to monitor Source 1, one of the drawbacks of the Object Monitoring Mode.

is employing outside sources which the system may not be monitoring. This leads to an incomplete picture of the users input. Since the system can only construct consolidations from what it can monitor, there is the probability that the consolidation is incomplete if not incorrect if the user is using other sources.

Subject Monitoring

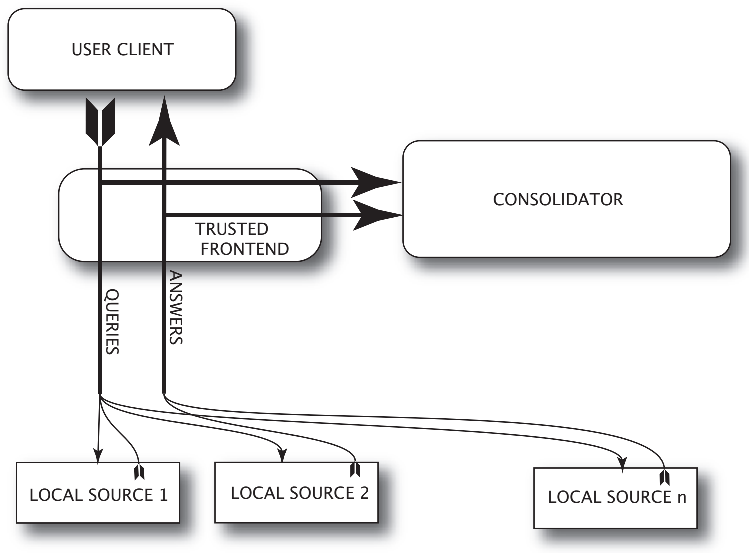


Figure 1.2: **Subject Monitoring Mode**

In the subject monitoring mode (Figure 1.2) the system will monitor the user's client (e.g., database browser or web browser). This mode is applicable for both security applications as well as user assistance and feedback applications. In the security case, the client with the monitoring system may be a trusted front end that any user is required to use while accessing data sources [2]. There are numerous advantages to this mode. First and foremost, user identification is trivial except for the case where several users collude to share the task, in order to mislead the system. Session identification is still a problem as the user will be looking for different things at different times. Incomplete information likewise becomes considerably easier to handle as the system can now see all the online sources the

[2]This is possible in organizations where it is policy to have employees use trusted browser/client software to access the intranet or the internet.

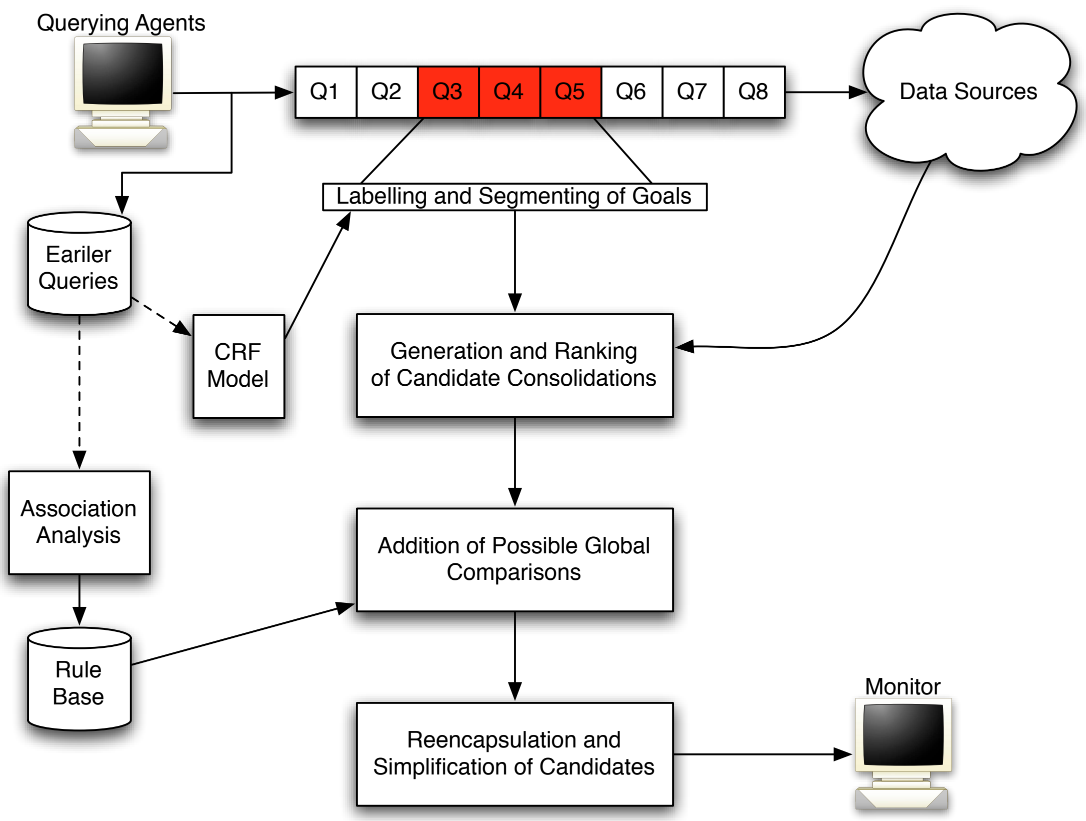

Querying Agents
Q1
Q2
Q3
Q4
Q5
Q6
Q7
Q8
Data Sources
Labelling and Segmenting of Goals
Eariler Queries
CRF Model
Generation and Ranking of Candidate Consolidations
Association Analysis
Addition of Possible Global Comparisons
Rule Base
Monitor
Reencapsulation and Simplification of Candidates

Figure 1.3: **Query Consolidation Workflow**

user is employing. Outside sources such as the user's common sense knowledge or domain knowledge is still missing but that is largely unavoidable.

1.6 Layout of the Dissertation

In order to describe the function of the subsequent chapters, I will first give a brief overview of the proposed approach. Figure 1.3 gives a schematic representation of the workflow for a possible query consolidation setup.

The query consolidation starts as with a stream of queries that arrive from one or more querying agents. For the rest of the dissertation only a single querying agent is considered without loss of generality. As will be apparent, the actions of different agents and the queries they ask may be considered independent and each agent can be monitored in parallel and completely exclusive of others. In a sense one can assume a "virtual" system monitoring one querying agent whereas in reality many of these virtual monitors may be operating on the same platform or distributed amongst several nodes.

The queries arriving[3] initially need to be packaged into meaningful tasks. This requires the processing of these streams of queries. Each packet is a set of queries that constitute, independent of the queries outside the packet, the components of a global query. I call each packet a "goal" in that they define one particular aim of the querying agent. These have been analogously called "sessions" in previous literature, particularly in works dealing with web usage mining. The approach to the problem of extracting these goals from a stream of queries uses some of the concepts and tools originating from the natural language processing field. Roughly, an analogy can be drawn between a stream of queries and natural language text, each query serving the role of a word. The self-contained goals are then analogous to sentences. The approach uses previously marked streams of queries to train a conditional random field model to effectively and efficiently mark the goal boundaries (i.e the query that is the beginning of each goal) in newly encountered streams. Furthermore, the same model can be used to label individual queries within the goal as to their function within that goal. This is again reminiscent of *part-of-speech tagging* in natural language processing. The detailed discussion of this method is given in Chapter 5.

Once a set of queries is extracted from the incoming stream one can attempt to find the most likely consolidation of these queries into a global query. The issue here, as previously mentioned, is that the query decomposition is not injective. Therefore, given a set of component queries there are multiple ways these may be assembled. I approach this problem by a variant of join inference. All the possible consolidations of a set of queries are ranked in order of the likelihood of the consolidation. This likelihood is a direct result of how "strong" the joins are among the constituent parts. I quantify this likelihood using an information theoretic measure based on the Kullback – Liebler divergence, or information

[3]or that have already arrived and that have been logged

gain, occurring on alternative joins. This join likelihood is assessed during both the labeling of streams and later in ranking the candidate consolidations. Since both these steps may be, depending on the application, operating in real-time (as the queries arrive), the join inference has to be very efficient. Hence, a less exact but very rapid approximation of this measure is also developed. The assumptions and methods used in generating the ranked consolidation candidates and these optimizations are described in Chapter 4.

Even after a ranking of consolidations has been generated, there are further improvements that can be done on the result. In the environment the consolidation is attempted, the monitor has only a limited view of the process by which the querying agent arrives at its global query. Even if all the local sources accessed can be monitored, there are certain operations that can be done at the integration site which are not visible to the monitor. These are operations performed by the querying agent only after all the pieces are joined and the resulting consolidation is filtered by further constraints. These constraints are "global" in that it would not be possible for the querying agent to perform them at the local sources. Hence, for the outside observer, there is no indication as to what constraints are being applied at the integration site. Therefore, a monitor can only *guess* at the global constraints that are likely applied by considering what has been retrieved. I approach this estimation using association analysis. Using earlier global queries as training data a rule base of associations can be created which relates a likelihood that the global constraint will be applied given the attributes that have been retrieved. This approach is discussed in Chapter 6.

Finally, once the candidate consolidations have been obtained, and possibly enhanced with possible global selections, there is an optional operation that can be performed to make the result more useful to a human observer. The final candidates are all sound in that they are equivalent to one of the global queries that could be assembled given a set of local queries. However, they are not necessarily as concise as possible. Parsimony is important in that it makes the intention behind a concise global query more apparent than a convoluted one. Furthermore, by the axiom of Occam's Razor, it can be argued that the simplest global query that covers the retrieved information is more likely to be the intended consolidation. I therefore describe a way to re-encapsulate the final answers in a simpler intensional description. One issue with a simpler re-encapsulation is that it may not be as precise as the original. This approach is further detailed in Chapter 7.

The layout of the dissertation is slightly rearranged so that the narrative develops in a more natural manner. The following chapter (Chp. 2) surveys the previous work relevant to the techniques and problems discussed in the remainder of this dissertation and therefore gives an idea about the areas of research within the scope.

Chapter 3 discusses the theoretical aspects of the research, establishes the theorems, bounds and assumptions inherent in the succeeding chapters and describes the model and environment the whole approach works under.

Chapter 4 explains the methods used to evaluate and rank the possible joins between components of a global query. After the concept of joinability has been thus elaborated, Chapter 5 returns to the beginning and discusses the methods used to extract individual goals from a continuous stream of queries and the incremental operation of the whole system.

Chapter 6, discusses the enrichment of the candidate solutions with the expansion of their projections and the addition of global selection constraints by using earlier experience.

Chapter 7 describes the optional re-encapsulation of the candidate solutions into simpler intentional representations using a genetic programming approach.

Chapter 8 gives experimental validation to all of the methods described above along with insights and discussions as to their scalability and the sensitivity of their accuracy noisy or incomplete input.

Finally, Chapter 9, concludes the dissertation with an overview of the technology, the results achieved, discussion of the shortcomings and possible future work.

Chapter 2: Background and Related Work

The subject of this dissertation is mainly rooted in the virtual database (Sec. 2.1) and query decomposition (Sec. 2.2) areas. I therefore start by giving a brief survey of the relevant work and basic definitions in these areas. Subsequently, I will go over several of the related subjects such as join inference (Sec. 2.3) and schema matching (Sec. 2.4) that are relevant to the methods used in this dissertation in discovering how pieces of a global query fit together.

In addition to the main subjects, the dissertation is based on the use of several technologies (i.e., 'tools') in attempting to achieve the main goals. One of these is association analysis (Sec. 2.5), which is used in finding supplementary global constraints to the global queries, once assembled. The sequence analysis and segmenting done on incoming query logs is inspired by session identification done in web log mining (Sec. 2.6) and makes use of conditional random fields (Sec. 2.7) in order to probabilistically label and segment the incoming queries, therefore the previous research and preliminaries in these areas will be briefly discussed as well. Finally, as the background for the re-encapsulation candidate results into simpler intentional expressions, I will go over the literature on intensional answering (Sec. 2.8) as well as a brief introduction to genetic programming (Sec. 2.9) which is used to generate simpler encapsulations.

2.1 Virtual Databases

Virtual Databases or information integration systems are distributed database systems which combine a heterogeneous collection of autonomous data sources. For virtual databases, the assembly site mentioned in the preceding section is the information integration system itself. The assembly site consists of a schema with no instance. The materialization of a global query is done by translating the query (with optimizations mentioned above where applicable) into multiple sub-queries to be materialized at the local sources and shipped back to the integrator for assembly [55].

Therefore, information integration systems are distributed databases in the general sense while having several special aspects and peculiarities not found in coordinated distributed databases. First of all, while not a strict rule, it can be contended that most distributed databases will have a star architecture, i.e., the integrator will have a connection to each local source and the local sources do not communicate with each other [21]. The second and primary concern for virtual databases is inconsistency. The inconsistencies possible between the components of an integration system can be classified as intensional or extensional [72].

Intensional inconsistencies, or schema inconsistencies, are the inherent semantic mismatches between the global schema of the integrator to the schemata or equivalent semantic representations of the local sources. One of the main tasks of the integration system is to reconcile these differences by mappings. There are three mapping paradigms applicable to schema mapping [55]:

Local-as-View (LAV): The mapping associates each relation of each local schema to a view of the global schema.

Global-as-View (GAV): The mapping associates each relation of the global schema to a view of a local schema.

Global-and-Local-as-View (GLAV): The mapping associates views of the global schema to views of the global schema.

There have been examples of integration systems employing each of these schema. The Information Manifold integration system [48] and a newer XML query based integrator reported in [65] are examples of LAV systems. TSIMMIS [36] is an example of a GAV approach. Multiplex is an example of a GLAV system [72].

Apart from how they handle their mappings, integration systems can further be classified based on the mechanism they use for their global schema. Some systems such as Multiplex and the Information Manifold keep static mappings from local sources to a fixed global schema. Other systems, such as TSIMMIS and HERMES [4, 5] are called “mediator”-based. Mediators are software constructs that handle semantic inconsistencies by using logical inference to reason about the incoming information and assemble ad-hoc knowledge elements. The drawback is that a new module must be developed for each new source. This is in line with the inherent problem of GAV, which makes it necessary to change the global schema as new local sources are added.

The second form of inconsistency encountered by an information integration system is extensional or data inconsistencies. These arise as different sources often have varied data on the same subject. There are various methods of resolving inconsistencies between data elements that have already been semantically reconciled. A straightforward way of handling data mismatches is giving all the different answers as a list. Other methods include voting [6] such that the consensus value for any element is return to the user.

The final alternative in literature to resolving inconsistencies is to fuse the answers into one true answer. Various ways of doing this have been proposed in literautre. TSIMMIS attempts fusion through its mediators which group the answers and try to logically combine them [75]. Multiplex does it at the record level by providing the largest consensus and smallest all-encompassing answers as lower (sound) and upper (complete) bounds for the true answer. The extension of Multiplex, called Fusionplex [73] attempts to fuse data at the element level using a multidimensional quality metric for each of the sources to do a weighted superposition of inconsistent data [74].

2.2 Query Decomposition

Query decomposition is a major issue in information integration, particularly for LAV and GLAV systems. In GAV integrators the problem is rather simple since each base relation in the global scheme is mapped onto a view on on of the local sources. If a global query requires a relation the part of the query relevant to that relation is simply nested with the view of the local source mapped to the relation. When there is a LAV setup however, the local source is a view on the global scheme so the problem becomes a problem of answering queries in terms of views which is a known difficult problem. In fact, finding the exact equivalent of a query in terms of arbitrary views is known to be undecidable [32]. A query Q_1 is said to be contained in another query Q_2 if the tuples retrieved by Q_1 are a subset of those returned by Q_2 for every possible instance of scheme. Q_1 is said to be maximally contained in Q_2 if there is no query Q_3 contained in Q_2 that also contains Q_1. Also, if any two queries Q_1 and Q_2 are contained reciprocally by each other they are said to be equivalent. Since the existence of this equivalence is undecidable, algorithms for answering queries in terms of views attempt to find a maximally contained answer for any given global query.

There are several algorithms in previous literature that handle the query decomposition in terms of views. The first of these is the *Bucket Algorithm* [56]. It is the first major algorithm described in literature and was developed for the aforementioned Information Manifold system. The algorithm can handle conjunctive views and conjunctive queries but not recursion. It does however support range queries (i.e., arithmetic comparison operators) and unions. The algorithm starts by considering each subgoal (i.e., conjuct) individually to determine the views relevant to the subgoal. For each such subgoal a "bucket" is created and the relevant views are placed in these. Subsequently, the algorithm checks combinations of elements from each bucket to see whether the combination is contained in the original query or if containment can be achieved by adding constraints. The approach is somewhat naïve in that it misses the interactions between conjuncts as it considers them independently. Therefore, it may end up considering many options that are destined to fail in the second phase.

Another approach to the problem of query decomposition is the Inverse Rules Algorithm [33]. The inverse rules algorithm can find maximally contained equivalents of both conjunctive and recursive queries in terms of conjunctive views. Although the method will be discussed further in Chapter 3, briefly the method works as follows: the view definitions are inverted such that the heads of the conjunctive view definitions become the bodies and the relations involved in the bodies become the heads of the inverted rules. Existential variables within the view definitions are Skolemized. The query is then simply evaluated in terms of the inverted rules. Any recursive or conjunctive datalog query can thus be answered. The Skolem functions introduced during inversion cause no problems since only the view definitions are simply conjunctive and never recursive. Therefore, the chaining of Skolem functions are guaranteed to be finite. The algorithm does not inherently support range queries or negation. An advantage of the method is that the inverted views can be generated once as local sources are discovered and recruited into the system and reused for subsequent queries. This makes the method technically dependent only on the complexity of the query and not the number of views. However, since the method does not check for semantic relevancy it may end up materializing many views that are not really relevant to the query, thus the real-world performance of the method may much worse than expected. As in the previous case, this algorithm was developed as part of a LAV information integration system, namely Infomaster. A second very similar but independently developed algorithm which only handles conjunctive queries is described in [81]

Finally, a more recent and improved algorithm called by MiniCon has been proposed by [80]. This algorithm was designed to improve the performance of earlier methods, and is particularly descended from the bucket algorithm, the aforementioned naïvete of which it fixes. Instead of considering individual subgoals independently as in the bucket algorithm, MiniCon considers the variables in the subgoals and their interaction with the local views. For every subgoal g in the query the algorithm looks at subgoals in the views and tries to find a match. When a matching subgoal g' in any view is found, the algorithm calculates what else would be needed in order to evaluate the query given that g is mapped to g'. For each match, this information is recorded into a data structure called a MiniCon Description (MCD) which therefore is essentially a containment mapping for that match. In the second part of the algorithm, the MCDs that are pairwise disjoint are combined into a single query which is guaranteed to be maximally contained in the original query. Like the bucket algorithm, MiniCon is able to decompose conjuctive queries over conjunctive views. The queries can be range queries and may have unions. Again, however, recursive queries are not suported. The Minicon algorithm does support limitations on variable binding however, as an improvement over the bucket algorithm.

For the remainder of this dissertation, I will focus on the Inverse Rules algorithm since it is simple, concise and natively supports recursive query plans.

2.3 Join Inference

Relational query languages such as SQL are well suited, in fact designed, to unambiguously retrieve a required set of tuples from databases of arbitrary size and complexity. However, by their very nature such languages are by no means intuitive and generally have a steep learning curve. Furthermore, it is not generally possible for the user to construct a query without knowing the schema of a database. Therefore, relational database systems and their associated query languages, while being undeniably useful, suffer from a lack of usability by the general public [26].

One of the main obstacles to usability can be identified as the difficulty of performing joins. Therefore, research has been ongoing, for the last 30 years, on systems that can infer these more complicated operations of query building without requiring the user to specify them explicitly.

Probably the first endeavour in this respect was the universal relation model [64]. The universal relation model attempts to make the joins among relations in a database transparent by automatically traversing the schema through join dependencies. There exists a problem of ambiguity, however when the underlying schema graph has cycles. In these cases there are multiple lossless join paths that can be constructed. The universal relation was thus extended with the concept of maximal objects [63] which are expert determined semantically meaningful partitions of the graph into acyclic subgraphs. Alternatively, these maximal objects can be generated by analyzing the functional dependencies that exist among attributes, again as described in [63]. In cases where a query covers parts of more than one maximal object, the union is given as the result.

Another approach to the problem of identifying the one join path expected by the user assumes the path with the lowest cost tends to be the right answer [91]. Here, the cost is computed by reducing the problem to a minimum directed cost Steiner tree problem and edge costs are defined in terms of the cardinality of the relationship.

Later approaches to the problem deviate from earlier ones in that they do not attempt to find a single correct join or merge all correct joins. Inspired by the way search engines work, newer methods tend to find several explanations as to how the join may be performed and rank them according to some relevance score which varies from study to study.

The Discover system, described in [42], uses keyword based queries. Once the keywords are located in the various relations of the database, these relations are connected through their primary-foreign key relationships. When several minimal join paths result, they are ranked according to the length of the join path, shorter join sequences having more relevance.

An improvement over this method was introduced in the DBXplorer system [11]. The essentials of DBXPlorer are the same as Discover except that it utilizes keyword indices[1] to more efficiently determine the locations of keywords in relations, whereas Discover scans relations each time.

Another query interface, INFER [66], generates and ranks the top-k join possibilities and allows the user to select the one intended before materializing the query. The results are ranked by prioritizing shorter join sequences over longer ones and lossless joins over lossy joins. The candidate generation in this system is particularly efficient, according to the authors, in that it precomputes the maximal join trees and looks up subtrees of these during query time.

The approach to join inference in this dissertation falls into a niche within the latter generation of methods, in that it ranks possible answers according to a relevance scheme. Due to the particular requirements of query consolidation it starts with a fixed number of relations to join. Ranking by join sequence length is not useful in this case since almost always the join sequences contain the same number of joins. Hence, the present ranking

[1]called *symbol tables*

scheme depends on the strength of the connections between various attributes of these relations and prioritizes spanning trees with higher total weights.

One property of this approach that distinguishes it from previous literature is that it is completely schema agnostic. In other words, the method can be given a set of relations to join without any indication as to which attributes are keys and what kind of relationships there are between these relations.

2.4 Schema Matching

Schema matching is a much researched area in information integration dealing with reconciling two schemas. The basic operation is the mapping of the elements of two database schemas to each other. Generally, in the context of information integration, one schema is the global schema of the virtual database and the other is that of a local source to be integrated. A comprehensive survey and classification of schema matching approaches are given in [83]. According to the classification done in the same work, schema matching approaches vary according to whether they use intensional (schema-level) or extensional (instance-level) information. In other words whether they match schemata according to only the metadata or using the actual data in the repositories defined.

There have been various instance-level schema matching approaches defined in previous literature. A good number of these are machine learning techniques to classify attributes as belonging to the same domain. Methods based on artificial neural networks [59], Bayesian inference with feature selection [17], meta-learning [31] and rule based systems [25] have been proposed. In the case of the problem in this dissertation schema matching methods are useful when a query cannot be mapped on to the existing global schema. In that case, one has to find an ad-hoc mapping between attributes in order to discover the relationship of one fragment with the rest. The data that is available (i.e., fragments to be joined) is expected to be specific to the instance at hand and not generally applicable. Therefore I approach the problem of finding domain compatibility between two attributes in one of two ways. In the exact (although slower) form of the method I make a closed world assumption and consider the instance at hand is all there is. Therefore while matching attributes' domains I use a straightforward overlap measure. For the approximate method where the whole of the data cannot be processed efficiently I use a constraint-based approach. I use a method of matching attribute domains based on the syntax of the values rather than the values themselves. This is a simplified version of the algorithm described in [76] for inferring pattern matching finite automata from examples of text patterns, which is in this case a sample of values for the an attribute. Such constraint-based approaches have been used in schema matching methods [54] and include data types, cardinality, range and syntax.

2.5 Association Analysis

Association analysis or association rule mining has been a considerably active field for more than a decade, since the initial treatment was proposed by Agrawal et al. [8]. Association analysis is mining a set of transactions in a database to find rules that generalize the associations among the items that constitute said transactions.

The major problem in association analysis has been the complexity of finding frequent item sets in a set of transactions. Assume that there is a catalogue of some N different items and a list of orders each of which contain $1 < m < N$ items together. While finding individual items occurring frequently in the orders is rather trivial, when all possible sets that can be built from these itmes are considered, the problem becomes very time consuming indeed. As a matter of fact, if each possible subset of the catalogue is to be considered, one would have to check $2^N - 1$ possible sets and count the occurrence of each one in the database of transactions. Clearly, such an endeavor would be virtually intractable for any case where N is respectably large.

Hence, a significant number of algorithms have been proposed which prune the search space into manageable proportions. The way almost all of these algorithms work is based on the fact that the frequency of occurrence of a set of items is anti-monotone with respect to its subsets. This frequency is termed the *support* of the set of items. If a set of k items is above a certain support threshold, by definition any of its subsets must also have at least the same amount of support. Hence, given a threshold for support, any set of items occurring at least as frequently as the threshold, along with its subsets, are termed *frequent itemsets*.

A related definition is that of *closed frequent itemsets*. Closed frequent itemsets are defined as those frequent itemsets which have no supersets with the same support. Note that, since support is anti-monotone, these supersets thus would have lower supports. Given a support threshold some of these closed frequent itemsets are such that all of their supersets have supports lower than the threshold. These are termed *maximal frequent itemsets*. In other words they are the largest sets of items occurring together in the database to the extent that they satisfy a minimum level of support. It is easy to see that the set of maximal frequent itemsets is a subset of the set of closed frequent itemsets which are in turn a subset of the set of frequent itemsets. Maximal frequent itemsets are also noteworthy in that they define the boundary in the search lattice where algorithms will prune all descendents for lack of support.

The methods suggested in the literature can be categorized into two pairs of orthogonal categories [41]:

- Breadth-first counting algorithms
- Breadth-first intersection algorithms
- Depth-first counting algorithms
- Depth-first intersection algorithms

I will briefly go over some exemplary algorithms that employ each of these approaches. The pioneering algorithm in literature, *Apriori* [10], is an example of a breadth-first counting algorithm. It is the first algorithm to utilize the anti-monotone property of support. Apriori works in a breadth-first manner, counting each level in one pass of the transaction database. So, all itemsets with k-1 items are counted for support before the first itemset with k items is counted. It is therefore possible to prune any k-item sets without counting them if any of their subsets are infrequent. Once these are pruned the remaining itemsets can be counted within the database.

A good example of a breadth-first intersection algorithm is reported by [84]. The advantage of the algorithm is that it can generate results in just two passes of the database and is thus suitable for very large databases. It divides the database into logical partitions such that each partition can fit in memory. Subsequently, each partition is counted separately to generate frequent itemsets for that partition only. The results are merged into a global list which may have false positives. These are then eliminated by a second pass over the whole database.

The depth-first search of a subset lattice is, at first glance, not such a good idea. Breadth-first searches count subsets at each cardinality level by doing one pass of the database. A depth-first search would require a counting pass over the database at each node, which, given the exponential number of subsets, would be terribly inefficient. However, several techniques and requirements vindicate a depth-first approach. Foremost, if one requires only the maximal frequent itemsets, depth-first analysis tends to be faster in finding the pruning boundary. Also, with maximal frequent itemsets look-aheads and neighbor branch pruning is also possible.[2] A good example of an algorithm that exploits these advantages, among others, is MAFIA [20].

Depth-first traversal is also aided a great deal by using transaction lists or bitmaps instead of direct counting. This way, at each node, the database does not require recounting, all that has to be done is the intersection of the parent node's list with the added element's list. Since in depth-first traversal only the current branch of the lattice is under consideration, the only lists or bitmaps that need to be stored in memory are those of nodes from the root down to the node currently under consideration. In contrast to breadth-first searching, where the whole level (i.e., all k-itemsets) has to be stored, in-memory counting by intersections is more viable in depth-first search [41].

A rather different approach, [39], deviates from the standard candidate set generation

[2]Given the set of items {A, B, C, D}, in a lexicographically ordered subset lattice, if one knows that {B, C, D} is maximal frequent, there is no longer a need to traverse subtrees rooted at {B, D}, {C} and {D}, since any descendants of these are already subsets of {B, C, D}. Likewise, if it is known that the nodes {A, B} and {A, B, C} have the same support, the node {A, B, D} does not need to be visited, since A, B and C obviously always co-occur and therefore there can not be a case where A and B exist with D and not C.

and pruning by employing a novel data structure called a Frequent Pattern Tree (FP-Tree). The building of this structure is done in a depth-first manner. The FP-tree is then used to compute the support values of frequent itemsets.

Generally, once frequent itemsets are found, association rule generation is done by dividing these itemsets into antecedent and consequents such that given the antecedent the consequent is probable to some extent, termed the *confidence* of the rule. The confidence is also bound by a minimum so an itemset $\mathbf{X}$ would be partitioned such that the rule $(\mathbf{X} - \mathbf{Y}) \rightarrow \mathbf{Y}$ has at least the minimum confidence where $\mathbf{Y}$ is a subset of $\mathbf{X}$. The use of association analysis in this dissertation somewhat at this point since, in the present problem, the antecedent is of a domain different than the consequent. Therefore, I modify the standard maximal frequent itemset generation as performed in standard algorithms such as those described in [7, 20, 39] slightly to fit the needs of the particular problem of query consolidation. This will be further elaborated in Chapter 6.

2.6 Session Identification

Most of the previous literature on session identification and analysis of query logs is geared towards web log mining. In web log mining, sequences of document requests from a web server are analyzed instead of database queries. A second form of web log analysis is the analysis of keyword and parameter queries sent to search engines and other information retrieval services (e.g., digital libraries, travel sites &c.) This type of analysis is based on clustering queries in order to find trends [14] and other clues as to the general information demands of the users in order to better design services [45].

In terms of session identification, the most prevalent and simplest method of isolating user sessions is the *timeout* method. In the timeout method, the user is assumed to have started a new task after a given amount of time has passed without activity. For web information retrieval the threshold is reported [40] to be 10 to 15 minutes, for optimal separation of subsequent user sessions. Another method proposed in [28] is based on the fact that the users spend less time on auxiliary pages (navigation pages &c.) than on content pages. The method assumes that a session is finished when a user navigates to content page and times out. The timeout method is applicable to the database query environment as well, as a supportive addition to the method proposed in the following section.

Finally, a method is proposed in [27] called *maximal forward reference*. In this method the user is assumed to be in the same session as long as the next page requested has a link from the current page. This approach is relevant to the problem at hand as it uses a metric of relevance between two requests to cluster them into the same session. A similar approach is used in the method proposed in the following section, to cluster relevant queries together.

As far as query log analysis in database environments, there is one relevant study [93] that uses methods based on statistical language analysis in order to do session identification. Their method is based on training a statistical model with previous examples. The method predicts session boundaries where a sequence of queries becomes improbable with respect to previous experience. The method differs from the one proposed by this dissertation in that it requires an in-order arrival. These concepts will be discussed further in Chapter 5.

2.7 Conditional Random Fields

The most common approach to segmenting sequence data and labeling the constituent parts has been the use of probabilistic graphical models. Particularly, *hidden Markov models* (HMMs) [82] have been used extensively in the literature for sequence data such as natural language text, speech, sensor logs and event streams. An HMM defines the joint probability distribution $p(X, Y)$, where X is a random variable over possible sequences of observations and Y is a random variable over the possible sequences of labels (i.e., the possible states of the process generating said observations) that can be assigned to the same input. Because they define a joint probability, HMMs are termed *generative* models.

A generative model can be used to generate new sequences obeying a certain distribution defined by the parameters of the model. These parameters are either coded directly or optimized by using training data. However, given a certain sequence of observations, finding the most likely sequence of labels involves using Bayes' Rule and knowledge of prior probabilities. Furthermore, defining a joint probability requires enumerating all possible sequences, both observations and labels. Given that the number of possible sequences increases exponentially with the sequence length, this is generally intractable unless strong assumptions are made about the independence of sequence elements and long-range dependencies thereof.

In contrast, a different class of graphical models, known as *discriminative models* model the conditional distribution $p(Y|x)$ for a particular sequence of observations, x and cannot generate the joint distributions. Since the problem at hand when trying to label a sequence of queries is finding the most probable sequence of labels given a sequence of observations (i.e., the observation sequence is already fixed), discriminative models are better suited to the task.

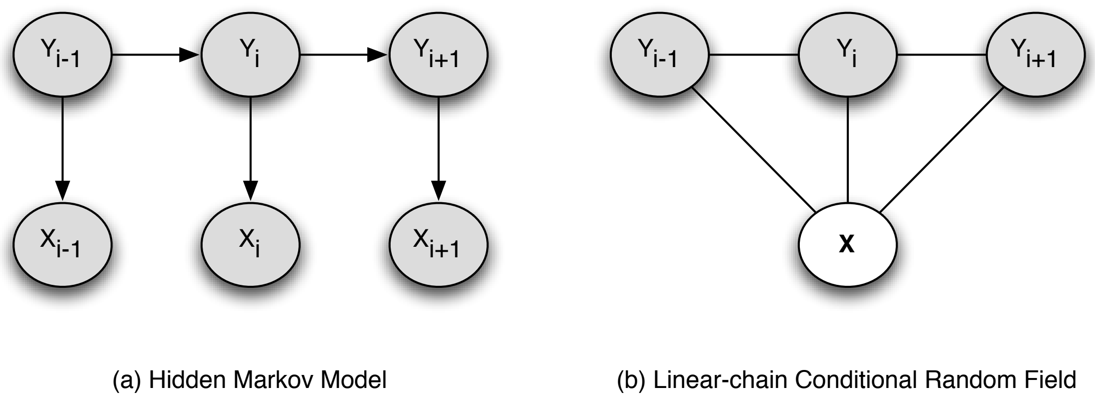


Figure 2.1: **Comparison of a Hidden Markov Model to a Linear Chain Conditional Random Field**

Shaded nodes indicate the variables that are generated by the model.

A rather recent form of discriminative model is the *Conditional Random Field* [53]. A conditional random field (CRF) can be regarded as an undirected graphical model, or Markov network [47], where each vertex represents a random variable. The edges of the graph represent dependencies between the variables. In the simplest case, the graph is a linear chain of state (label) variables Y_i dependent on the previous state and globally conditioned on the sequence of observations X (see Fig. 2.1b) but higher order and arbitrary graphs are possible. As long as each state variable Y_i obeys the Markov property (i.e., is conditionally dependent only on its clique in the graph), the graph is a conditional random field.

Given this structure, it is therefore possible to factorize the joint distribution of Y into potential functions, each one contained to a clique of vertices in the CRF. In other words, the arguments of each potential function must be limited to the same clique of vertices. In the case of a linear chain CRF such as that given in Fig. 2.1, the arguments of any such function will be Y_i, Y_{i-1}, and X. As long as the feature functions are positive and real valued, the product will satisfy the axioms of probability, provided that it is normalized by a factor. Therefore the definition of any potential function f_k for a clique consisting of a particular observation sequence $\mathbf{x}$ and elements y_i and y_{i-1} of a particular state sequence $\mathbf{y}$ would be:

$$f_k(y_i, y_{i-1}, \mathbf{x}, i) = r \in \mathbb{R}, 0 \leq r \leq 1$$

The values of these potential functions can then be aggregated over the complete sequence to obtain the feature functions:

$$F_k(\mathbf{y}, \mathbf{x}) = \sum_{i=1}^{n} f_k(y_i, y_{i-1}, \mathbf{x}, i)$$

In trying to segment and label a sequence of queries, the interest is only in *decoding* a particular state sequence $\mathbf{y}$. In other words, given a particular sequence of observations one would like to find out the sequence $\mathbf{y}$ that would maximize $p(\mathbf{y}|\mathbf{x})$. This conditional distribution associated with the CRF is:

$$p(\mathbf{y}|\mathbf{x}, \lambda) = \frac{1}{Z(\mathbf{x})} \exp \sum_{k=1}^{K} \lambda_k F_k(\mathbf{y}, \mathbf{x}) \tag{2.1}$$

where $Z(x)$ is the aforementioned normalization factor, and λ is the set of parameters (or weights) associated with each of the K feature functions. For any given application the training of a CRF involves estimating these parameters. This parameter estimation is done using maximum likelihood training. Given a set of training sequences consisting of an observation sequence and the associated label sequence, $\{(\mathbf{x}^{(j)}, \mathbf{y}^{(j)})\}$, the product of Eqn. 2.1 over all the j pairs of sequences is the likelihood. In maximum likelihood training the set λ that maximizes the log-likelihood function:

$$\mathcal{L}(\lambda) = \sum_j \left[\log \frac{1}{Z(\mathbf{x}^{(j)})} + \sum_k \lambda_k F_k(\mathbf{y}^{(j)}, \mathbf{x}^{(j)}) \right] \tag{2.2}$$

is found. The log-likelihood function is concave and differentiable. However, the equating the differential to zero does not necessarily result in a closed-from in terms of the parameters. Therefore, the parameters that maximize the likelihood cannot be solved for analytically [92]. However, the problem is amenable to numerical solution by iterative scaling [18, 44] or quasi-Newton methods [60].

2.8 Intensional Answers

The foundations of the research on qeury re-encapsulation in this dissertation can be traced into previous literature along two primary headings. The first is the previous state of the art in intensional answering. The second heading consists of genetic programming and especially genetic programming as it applies to databases and data mining. As far as I know, this dissertation is the only point of intersection for the two areas, therefore the subjects are thus far necessarily unrelated. To this end I will go through them independently, starting with the former.

Intensional answering arises as a form of cooperative database function in which system returns a description of the result as a set of rules or predicates instead of a set of tuples. The thinking here is that rules are conceptually more informative and easier to comprehend to the user than a set of derived tuples [43]. The results of any query may therefore be seen as either an information centric *extensional* answer or a knowledge centric *intensional answer*. In other words, a set of tuples that comprise a view of the database is the extension and the predication that defines that view is the intension [71]. These two forms of answers are possible exclusively or in a mixed manner. The choice of answer depends mostly on the query itself. If the user is asking for a list of phone numbers, an intensional answer is almost useless. On the other hand, a query looking for employees making more than a specified amount is most likely looking for a class of employees rather than a list of names. An answer such as "Managers and Engineers make more than X" might be appreciated more by the user. One aspect that is generally of concern is that the intensional answer itself should be considerably more concise than the extension [85]. It is easy to see from the previous example that a predicate enumerating all the people who make more than X is necessarily the same thing as returning the extensional answer.

There have been several ways in which previous literature has attempted to handle intensional answering. These include using the integrity constraints of the database [70], in which every query is executed normally on the data to obtain the extensional answer, but also accompanied with an intensional answer derived from the schema and integrity constraint metadata associated with the database. Another work has proposed using decision tree methods to classify the extensional answer as a class in the whole database and subsequently derive the intensional predicate from the resulting decision tree [16]. This is closer to the approach in this dissertation in that it uses machine learning techniques to classify the set of tuples in the answer to derive the intensional answer. Totally different methods involving logic programming [30] and deduction [79] have also been suggested.

2.9 Genetic Programming

Genetic programming [29] has developed over the last two decades as local-optimization method for generating simple computer programs that present solutions for semi-structured or "black-box" problems. The point to notice is that these programs are *evolved* to map to the required values of a range given distinct values of the domain. The extent to which the programs can map to the correct results defines their *fitness* to the problem. The programs which have the highest fitness survive as the optimization progresses. These programs can thus recombine with other such programs to generate offspring of even higher fitness. The ultimate goal is a program which can attain performance above a certain threshold fitness.

The basic principals of genetic programming was defined by the landmark book by J. Koza [50]. This publication forms the foundations of genetic programming, including the basic tree generation algorithms, breeding operations and the basic evolutionary cycle used by my approach. A second publication by the author [51] is also relevant to this dissertation in that it outlines mechanisms to use the optimal substructure properties of the problem and defines reusable program portions called *automatically defined functions* (ADFs). The atomic selection operators, described in Section 7.2, are essentially ADFs and my approach tries to exploit optimal substructure by starting from very simple sub-queries and build an accurate but concise intensional answer in a bottom-up fashion.

Genetic programming has been used to some extent in data mining. In some cases researchers have preferred genetic programming as a classification method instad of or along with more traditional methods such as decision trees. The reason is generally to increase the accuracy of prediction. Genetic algorithms are better able to capture trends and rules in smaller subsets of the dataset which more conventional methods usually overlook [23,24]. Genetic programming and algorithms have been successful enough in data mining to prompt complete systems which have been built on the premise and commericalized [34]. For a more complete treatment of genetic programming and genetic algorithms in data mining see the survey by Freitas [35].

Chapter 3: Preliminaries and Theoretical Results

In this chapter, I will start by making the basic definitions used throughout the dissertation and also introduce notational conventions used throughout. Subsequently, I will describe the data model. Since query consolidation is the reverse of information integration, the consolidator uses a conceptual integration scheme as a substrate. This substrate in effect defines the knowledge of the consolidator on the domain. I will go on to define the query decomposition method used as a benchmark. Using the behavior of the decomposition algorithm I will introduce the primary assumptions used in the rest of the research. Finally, I will introduce the theoretical framework based on this model and assumptions, discussing the possibilities of consolidation and specifically the combinatorial problems that result from the multiplicity of correct solutions given a consolidation problem.

3.1 Basic Definitions and Notation

The environment in which I study the query consolidation problem is the relational model. Thus the most basic data structure involved will be the relation. A relation, in its simplest sense, defines a connection between one or more objects, concepts, classes or, in general, attributes. Relations will be denoted by uppercase letters (e.g., R, S, T) or capitalized short phrases that define their nature (e.g., Flight, Employee, &c.). Base relations in schemata are generally named with letters starting from R, views are named with letters starting from V. Relations defined by queries are denoted with Q, with appropriate subscripts when there are more than one query in question. When required for clarity, the relations may have their schemata listed along with their names (e.g., R(A,B,C)). Individual attributes will be denoted with uppercase letters starting with A, or may be denoted as capitalized short phrases defining the attribute. Sets of attributes will be denoted in boldface capital letters starting with $\mathbf{X}$ (e.g., $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$).

There is a certain duality when considering relations which will become very important. A relation may be regarded as an intensional concept, in which case the *sense* of the relation is meant. Otherwise the relation may be thought of as an extensional entity, in which case the relation is simply a set that contains all the tuples that obey the intension of the relation. Notationally, when the relation is mentioned in its intensional sense I will use standard uppercase or capitalized text. For the extensional form of relations, since they are sets of tuples, I will use a boldface notation (e.g., R: colors of the rainbow, $\mathbf{R}$ ={ (Red), (Orange), (Yellow), ... , (Violet)}.). Relations can thus be intensional or extensional, with the extensional form being an instance of the intensional form, i.e., sch($\mathbf{R}$) = R.

Database schemata can be simply regarded as collections of base relations. For the rest of the dissertation, whenever mentioned, schemata will be denoted with a calligraphic typeface (e.g., $\mathcal{G}$ for a global schema). Similar to individual relations, the instance of the same database will be denoted in boldface (i.e., $sch(\mathbf{G}) = \mathcal{G}$).

In this research the scope is limited to conjunctive views. A conjunctive view over a database is defined as the result of a conjunctive expression on the database. A formal definition is as follows:

Definition 3.1 (Conjunctive View)**.** A view V over a database $\mathcal{D}$ is defined by a Horn clause of the form:

$$V(\mathbf{X}) \quad \leftarrow \quad R_1(\mathbf{Y_1}), \ldots, R_n(\mathbf{Y_n})$$

where $R_{1 \ldots n} \in \mathcal{D}$ and $\mathbf{X} \subseteq (\mathbf{Y_1} \cup \ldots \cup \mathbf{Y_2})$. In other words, the predicates should all be base relations of the database $\mathcal{D}$ and the all variables in the head of the clause should appear in the body as well. Similar to all relations, the materialization (i.e., extension) of the view V is denoted as $\mathbf{V}$ □

Queries are regarded as "ad-hoc" views. The formal definition of a query is the same as that of a view. In practical terms, views are assumed to be more or less persistent over the life of the database while queries are specific to a single information request. In both queries and views, an important definition is that of containment.

Definition 3.2 (View Containment)**.** Given views V and U over a database $\mathcal{D}$:

$$V \subseteq U \Longleftrightarrow \mathbf{D} \models \mathbf{V} \subseteq \mathbf{U}$$

Therefore, the view **V** is contained in **U** if its extension, V is a subset of the extension U, for all instances of the database. □

The definition of containment leads to the definition of view equivalence. Two views are equivalent if they contain each other:

Definition 3.3 (View Equivalence)**.** Given views V and U over a database $\mathcal{D}$:

$$V \equiv U \Longleftrightarrow (V \subseteq U) \wedge (U \subseteq V)$$

□

3.2 The Data Model

Since query consolidation is ultimately the reverse of data integration, one needs a virtual database model that will be the substrate on which the monitor operates. As reviewed in the previous chapter, there are many different virtual database architectures that have been proposed in previous literature. These ultimately represent different variations on three basic approaches to the problem of integrating data, namely Local-as-View (LAV), Global-as-View (GAV) and to a lesser extent, the hybrid of these two, Global-Local-as-View (GLAV).

In this dissertation, I adopt the Multiplex architecture [72] as a starting point. The advantages of Multiplex that are attractive here are its simplicity and generality. Simplicity is due to the fact that Multiplex assumes that all databases are in the well-known relational model, without introducing any new concepts or structures.[1] Generality is achieved by the method in which the global and component (local) databases are associated, namely as arbitrary view pairs, hence making it a GLAV system.

A Multiplex database is

1. A global database scheme $\mathcal{G}$,
2. A set $\{\mathcal{L}_1, \ldots, \mathcal{L}_n\}$ of local database schemes, and their associated database instances $\mathbf{L}_1, \ldots, \mathbf{L}_n$, and
3. A set $\mathbf{M} = \{(V_1, U_1), \ldots, (V_m, U_m)\}$ of view pairs, where each V_i is a view of the global scheme $\mathcal{G}$, and each U_i is a view of one of the local schemes.

The global scheme is defined in the relational model in Multiplex. Hence, it consists of a set of relations. A portion of these relations will encapsulate the real-world concepts in the application domain (e.g., employee, department, project &c.). Other relations hold the relationships of these concepts to each other (e.g., WorksIn, Manages &c.). In addition to the relations themselves, a set of known generalized dependencies can be encoded in the scheme.

Definition 3.4 (Generalized Dependency)**.** A generalized dependency [38] is an implication of the form:

$$\forall \bar{X} [\phi(\bar{X}) \Rightarrow \psi \bar{Y}]$$

Where ϕ and ψ are conjunctive expressions of relations and equality assertions on variables $\bar{X}$ and $\bar{Y}$ respectively. $\square$

This form of generalized dependency encapsulates all of the relevant dependencies one may want to capture in the scheme. The most important being functional dependencies,

[1]Yet, as noted in [72], with certain assumptions, local databases could be in other models as well.

all other equality and tuple generating dependencies such as join dependencies and multi-valued dependencies are covered. For example, a functional dependency such as $A \rightarrow B$ on a relation R may be written as:

$$\forall A \forall B \forall B'[R(A,B) \wedge R(A,B') \Rightarrow B = B']$$

Inclusion dependencies are enforced only during data entry and updates. Since neither the querying agent nor the monitor have much control over the quality of the data in the local sources, it makes no sense enforce these extra constraints. Note that functional dependencies can be violated as well, due to duplicates with different values and extensional inconsistency between the local sources. However, since functional dependencies encode the essential relationship information between attributes and are thus included in the global scheme.

Finally, the global scheme includes a set of mappings between attributes. Reminiscent of schema matching mechanisms, this set contains a mapping between two attributes (e.g., R.A and Q.B) if the attributes have the same meaning.

Example 3.1. Consider a schema consisting of two relations: $\mathit{Dept}(ID, Name, Chair)$ and $\mathit{Faculty}(ID, Name, \mathit{Rank}, \mathit{Dept})$. In this case DeptID in the Dept relation and the Dept attribute in Faculty refer to the same concept, namely the identifier of a department. Likewise Dept.Chair and Faculty.ID both refer to the ID number of a faculty member. Thus, the mapping set for this example becomes $\Phi = \{< \mathit{Dept.ID}, \mathit{Faculty.Dept} >, < Faculty.ID, \mathit{Dept.}Chair\}$. $\square$

Formally, then, a global database scheme $\mathcal{G}$ is a triple $< \mathbf{R}, \mathbf{\Sigma}, \mathbf{\Phi} >$ composed of:

- A set of relations $\mathbf{R}$ and
- A set of generalized dependencies $\mathbf{\Sigma}$
- A set of pairs Φ mapping the real world meanings of attributes to each other.

This is the bare minimum of information needed to build the global scheme. Note that one can readily derive key information from the closure $\mathbf{\Sigma}^+$. Given a relation R, any functional dependency of the form $X \rightarrow R$ indicates that X is a key of R. Likewise, given $X \subseteq R$ and $Y \subseteq Q$, the following conclusions can be drawn for cases where dom(X) = dom(Y):

- If $X \rightarrow R$, R has a 1-to-many relationship with Q.
- If $Y \rightarrow Q$, Q has a 1-to-many relationship with R.
- If both $X \rightarrow R$ and $Y \rightarrow Q$, Q and R have a 1-to-1 relationship.

The global scheme $\mathcal{G}$ has no instance. It serves as a template while materializing the answers to queries posed by the users of Multiplex. A view pair (V_i, U_i) is a mapping between the local data source $\mathcal{L}_i$ and the global scheme $\mathcal{G}$.

Having a mapping between a global view V_i and a local view U_i implies that these two views are equivalent. In other words, for any possible instance $\mathbf{L}_i$ of a local database $\mathcal{L}_i$ the extensions of the local view U_i would be equivalent to the extension of the global view V_i, had there been an instance of the global database $\mathcal{G}$.

This is where the GLAV property of Multiplex becomes evident. Notice that if all views V_i were equivalent to individual base relations of the global database, each local view U_i would be mapped directly on to a single relation, thus making it a global-as-View architecture. Alternatively, if each local view U_i was mapped one-to-one on individual relations of the local databases, the mapping is reduced to a local-as-View architecture.

When a query is posed to the Multiplex system, the relevant parts of the global scheme are considered and the query is rewritten in terms of the views V_i that overlap with these parts. The resulting queries are sent to the local databases to be evaluated against the views U_i of those local databases.

3.3 Query Decomposition

As mentioned in the previous chapter, there are several query decomposition algorithms in previous literature. Of these, I have chosen to base further analysis of the problem on the Inverse Rules algorithm described in [32, 33]. All the algorithms reviewed previously take the same basic approach to answering queries in terms of views and the Inverse Rules algorithm was chosen because it was found to be the most succinct and least ambiguous treatment of the subject. That being said, the analysis performed here is applicable to any query decomposition algorithm as long is it is correct and efficient. Inverse Rules also guarantees maximal containment of the answer in the original intension of the query.

Maximal Containment is defined in terms of a global query Q, an assembly expression $E(V_1 \ldots V_n)$ and a set of views $V_1 \ldots V_n$.

Definition 3.5 (Maximal Containment)**.** A query plan E is maximally contained in a query Q (i.e., $Q \supseteqq E$) if for all possible instances of $V_1 \ldots V_n$:

$$E \subseteq Q \wedge \nexists E' : (E' \subseteq Q \wedge E \subseteq E')$$

To recap how Inverse Rules works, consider a simple example:

Example 3.2. Let the information integration system consist of a single relation of the form:

$$Flights :< From, To, FlightNo >$$

and be connected to two local sources L_1 and L_2 with the mappings

$$V1 :< To, FlightNo > \quad \Leftrightarrow \quad Flights(`IAD', To, FlightNo)$$
$$V2 :< To, FlightNo > \quad \Leftrightarrow \quad Flights(`DCA', To, FlightNo)$$

respectively. Therefore, the local sources map to the same base relation on the global schema with different constraints, i.e., one is the outgoing flights from Dulles Airport and the other from Reagan Airport. Initially, the inverse rules algorithm prepares an inverted set of views as rules. The inversion is such that the head of the rule is a base relation and the body is in terms of views. The two inverse rules would be:

$$\begin{aligned} Flights(`IAD', To, FlightNo) &:- \quad V1(X, Y) \\ Flights(`DCA', To, FlightNo) &:- \quad V2(X, Y) \end{aligned}$$

Now, assume that the query is:

$$Q(To, FlightNo) : -Flights(To, `JFK', FlightNo)$$

namely, the flights to JFK. The inverse rules algorithm simply evaluates the original query along with the inverse rules, so that the query plan becomes:

$$\begin{aligned} Q(To, FlightNo) &:- \quad Flights(From, `JFK', FlightNo) \\ Flights(`IAD', To, FlightNo) &:- \quad V1(To, FlightNo) \\ Flights(`DCA', To, FlightNo) &:- \quad V2(To, FlightNo) \end{aligned}$$

Note that the list of flights obtained from this query plan is not going to be complete. However, since the system is linked to only two airports it will retrieve the maximally contained answer, i.e., the list of flights to JFK, given the available information. □

For views that contain a conjunction of multiple base relations (i.e., joins), the reversal involves introducing Skolem functions.

Example 3.3. Starting off with the database of the previous example, let there now be a third mapping[2].

[2] In order to keep the example simple, assume that a given airline has only one flight between two locations.

$V3 :< To, From, Airline > \Leftrightarrow Flights(To, From, FlightNo), Operator(FlightNo, Airline)$

The set of inverse rules now becomes:

$$\begin{aligned}
Flights(`IAD', To, FlightNo) &\ :- \ V1(To, FlightNo) \\
Flights(`DCA', To, FlightNo) &\ :- \ V2(To, FlightNo) \\
Flights(To, From, f_1(To, From, Airline)) &\ :- \ V3(To, From, Airline) \\
Operator(f_2(To, From, Airline), Airline) &\ :- \ V3(To, From, Airline)
\end{aligned}$$

□

Notice that the function terms can be treated as regular variables and become bound when all of their arguments are bound as well. Since the function terms are only introduced during inversion of view definitions, and since the view definitions are conjunctive, the functions are guaranteed to not have nested functions as arguments. They do not therefore prevent the evaluation of any query plan and can indeed be eliminated altogether as explained in [33].

The inverse rules algorithm also naturally supports recursive query plans. Recursive plans are of great importance since even if they are not explicitly used, users sometimes are forced into recursive behavior in order to collect the information they need, as will be seen further on.

Example 3.4. Consider the case where the query is:

$$\begin{aligned}
Q(X, Y) &\ :- \ Flights(X, `JFK', Y) \\
Q(X, Y) &\ :- \ Flights(X, Z, Y), Q(Z, W)
\end{aligned}$$

This is a recursive plan that searches for any flights either direct or connecting flights to JFK. This query may be combined with the inverse rules from Example 3.3 to obtain a query plan that will retrieve all known flights to JFK, direct or connecting. □

3.4 Assumptions

I will make two primary limiting assumptions that will define a baseline for the rest of the research. While the former of these will be relaxed in certain points when possible, they are nevertheless central to the decisions made hereafter.

3.4.1 Language Assumptions

The first assumption on the expressiveness of the query language. I assume that all the queries and view definitions are in Selection-Projection-Cartesian product (SPC) relational algebra [3]. Although SPC is a restricted set of relational algebra (i.e., it excludes union, difference, non-conjunctive selections, and aggregations), it is adequately expressive for the large portion of queries used in the real world [57].

The simplification this assumption allows is that one can now use a canonical expression for all queries and views expressed in SPC. It can be shown that any expression in a SPC algebra can be written in the form[4] :

$$Q = \pi_{\mathbf{P}} \sigma_C (R_1 \times R_2 \times \ldots \times R_n) \tag{3.1}$$

Therefore, each query or view may be defined by a tuple $Q = \{\mathbf{P}_Q, C_Q, \mathcal{R_Q}\}$ where $\mathcal{R_Q}$ is the set of relations mentioned in Q. $\mathbf{P_Q}$ and C_Q are the projection attribute set and selection condition expression of Q, respectively.

[3]Or, equivalently, Selection-Projection-Join-Rename (SPJR), in the named perspective

[4]See [1] for proof.

The selection condition C_Q is defined as a conjunctive predicate with each term being a comparison between two variables (i.e., attribute names) or a variable and a constant. Comparisons between constants and variables are termed *simple comparisons* and comparisons between two variables are termed *immediate comparisons.*

3.4.2 Behavior Assumptions

As stated in in the introduction, the main problem is:

> Given local queries $Q_1, \ldots, Q_n$, find a global query Q and an expression E such that $Q \supseteqq E(Q_1, \ldots, Q_n)$.

One can think of the problem as a reversal. I start by assuming that there is a "querying agent" that is sending out queries to several sources in order to collect information which will in turn answer a global question. The aim is to infer this question by looking at queries of the agent, which is assumed to be rational and efficient.

There are certain assumptions inherent in considering a rational and efficient querying agent. The first is the *necessity assumption.* Here, I assume that the local queries are such that they only project necessary attributes. Likewise, any queries that are not necessary for the integration would not be asked. Therefore, $Q_1 \ldots Q_n$ are assumed to be *necessary* to build Q.

Recall that query decomposition was characterized as a procedure with a unique outcome. Consider a simple global query that retrieves a single value such as a person's age. Obviously, there could be multiple correct decompositions. For example, the local query Q_i could retrieve just the person's age; or it could retrieve that person's entire tuple, and the expression E would project the age; or it could retrieve the tuples of multiple persons and E would select that person's tuple and project the age. The guiding principle of the query decomposition procedure is to retrieve from the local databases as little as possible, taking advantage of the local system's query processing capabilities. This reduces possible costs charged by the local database, as well as the costs and time of transmitting the data. Hence, the decomposition adopted is one that *optimizes* the process.

An important dimension of the necessity assumption is that it restricts the the degrees of freedom on C_E due to frugality in processing. This frugality assumption states that any information that can be filtered locally will be filtered locally. This means that C_E cannot contain any selection terms that are relative to constants (i.e., simple comparisons). This is better illustrated by an example:

Example 3.5. Given $Q = \pi_A \sigma_{A=3}(V_1)$, where $V_1 = \{A, B, C\}$ is the global view of the local database.
Consider these two formulations of E:

1. $Q = E(Q_1) = \pi_A(Q_1)$, where $Q_1 = \pi_A \sigma_{A=3}(V_1)$

2. $Q = E'(Q_1') = \pi_A \sigma_{A=3}(Q_1')$, where $Q_1' = V_1$

The second formulation (i.e., E') is flawed since Q_1' is not the most frugal local query strategy. In this formulation all the tuples in local database D_1 are retrieved whereas Q_1 retrieves only column A and only the tuples where $A = 3$.

Through the standard transformation rules of relational algebra, any constant-based selections can be pushed into the local query [1]. Therefore, the terms of C_E are assumed to only contain immediate comparisons (e.g., $A = B$, $A > B$, &c.). Furthermore, any immediate comparisons that can be done within a local source will be done at the local side. So C_E only contains immediate comparisons between variables originating from different local sources.

The second assumption is that of *sufficiency.* This assumes that the user is not using information obtained elsewhere to achieve his goal. The information in the local queries $Q_1, \ldots, Q_n$ is sufficient to achieve the goal, and hence can be approximated by an appropriate consolidation.

These two assumptions restrict the problem space considerably. They are, however, not justified in some cases. The necessity and sufficiency assumptions are based on the more general assumption that the querying agent is both perfectly logical and honest. Note that by “querying agent” I refer to both automated integration systems as well as a human operator manually collecting and integrating data.

If the integrator makes logical errors, it may well ask for attributes or even whole queries that are not going to be used in the global question. Alternatively, even a logical integrator may ask for spurious data in an attempt to misdirect any surveillance. Likewise, in conflict with the frugality assumption, selections that could be done locally could be instead done more expensively at the integration site as a result of poor optimization or a purposeful attempt to mislead.

Mindful of these considerations, I still adopt these two assumptions in order to be able to approach the problem in a simple manner.

3.5 Analysis of the Extent of Solution Multiplicity in Consolidation

One may start the analysis of the non-unique nature of query consolidation from the point of view of the querying agent. Any conjunctive query can be put into a canonical form. From a relational algebra point of view, a conjunctive query is any query expressible in the project-select-join algebra. The canonical form for any such query is:

$$Q = \Pi_{\mathbf{P}} \sigma_C (R_1 \times R_2 \times ... \times R_n)$$

where $R_1 ... R_n$ are the base relations used, $\mathbf{P}$ is the set of projection attributes and C is a conjunctive predicate. The predicate C is by definition a conjunction of terms of the form $A_i = A_j$ or $A_i = K$ where $A_{i,j}$ are attributes of the cartesian product and K is a constant value. $A_i = A_j$ type terms are essentially how one performs equi-joins of the component relations and $A_i = K$ type terms are how the results are filtered to include only the desired tuples for the query. Note that the comparison operators used may not be limited to equality and may be any binary comparison such as arithmetic comparison, lexicographic ordering, string length &c. This being the case for all SPJ queries, a global query decomposed by a query decomposition algorithm may thus be written as:

$$Q_G = \Pi_{\mathbf{P}}\sigma_C(V_1 \times ... \times V_n)$$

where $V_1...V_n$ are the global views used. By the rules of relational algebra, this form can be re-expressed as:

$$Q_G = \Pi_{\mathbf{P}_G}\sigma_{C_G}[\Pi_{P_1}(\sigma_{C_1}(V_1)) \times ... \times \Pi_{P_n}(\sigma_{C_n}(V_n))] \tag{3.2}$$

by distributing the projection and selection operators over the cartesian product. In fact, this would be exactly what a query optimizer would do in order to push as much of the processing down to the local sources and to minimize the amount of intermediate results to be transferred and processed. Simply by replacing the global view V_i with the local view on the source, U_i, and maybe changing certain attribute names and constant values with respect to the rules of the mapping, the query decomposition algorithm arrives at the set of local queries to be sent to the local sources. These would therefore be of the form:

$$Q_i = \Pi_{\mathbf{P}_i}(\sigma_{C_i}(U_i)) \tag{3.3}$$

Therefore the relationship between the global query, Q_G, and its components, Q_i, (i.e., queries decomposed to local sources) is:

$$Q_G = \Pi_{\mathbf{P}_G}\sigma_{C_G}[Q_1 \times ... \times Q_n] \tag{3.4}$$

The model of a querying agent is that of a rational and efficient one. Therefore I assume that the querying agent will optimize its queries as much as possible. This would mean that any constant comparisons (i.e., of the form $A_i = K$) will be processed at the local source if possible. Therefore if a local source, D_i, can make the selection against an attribute the query Q_i sent to that source will have the required selection predicate in its conjuction

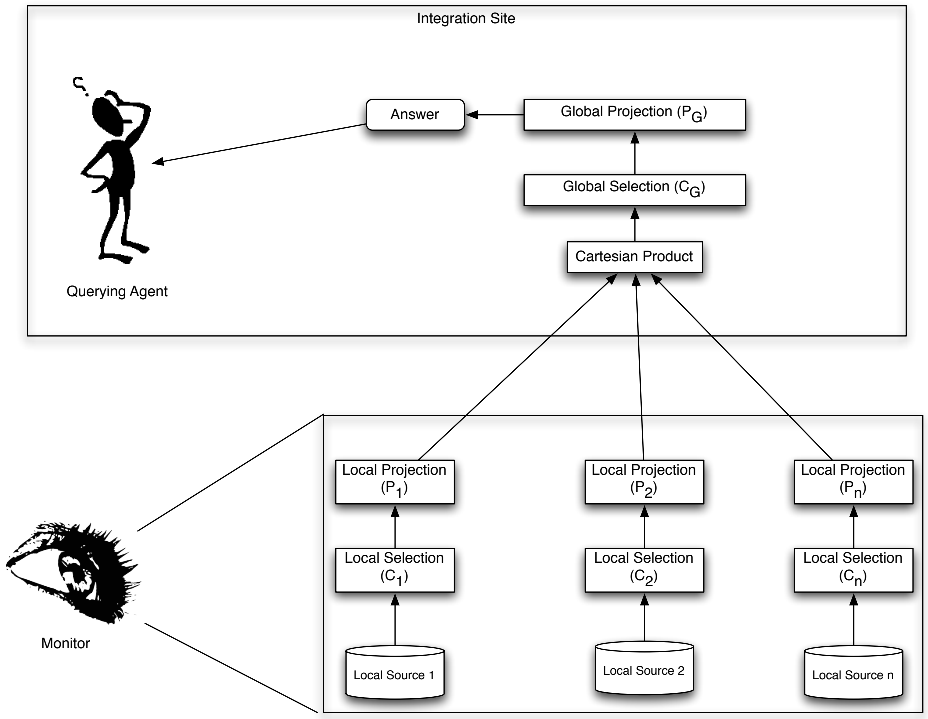


Figure 3.1: **Visibility of the Query Process**

C_i. Likewise, if some of the attributes in the view for a source are unnecessary for the computation of the global query these are assumed to be projected out at the local source (i.e., using $\mathbf{P}_i$) rather than at the integration site (i.e., using $\mathbf{P}_G$).

Looking at the operation from the point of view of the monitor (Figure 3.1), the only visible entities are the component queries, Q_i, that the querying agent sends out to local sources. The monitor has to guess at the global query, Q_G, by considering these. The missing information are the operations done at the integration site by the querying agent. For the case of PSJ queries, these are the contents of $\mathbf{P}_G$ and C_G.

In order to estimate an upper bound on the number of possible solutions, I start with the worst case scenario. Consider the following:

Lemma 3.1. Let a querying agent retrieve N relations from different sources. Each relation has an arity of M attributes.

The worst case in the size of the consolidation space is when each attribute of each retrieved relation is of the same type (e.g., all integers), and therefore comparable to each other in terms of joins or global selections. □

As a very simple example to the preceding lemma, consider the following:

Example 3.6. Given two components $Q_1 :< A, B >$ and $Q_2 :< C, D >$

$\mathbf{Q}_1$:

A	**B**
1	2
2	4
3	6
4	8
5	10

$\mathbf{Q}_2$:

C	**D**
1	1
5	2
10	3
15	5
20	6

Even when the projection is assumed to be the union of the projection sets of Q_1 and Q_2, and with just one possible operation (i.e., equality of attributes or lack thereof), there are four possible *minimal*[5] consolidations:

$\sigma_{A=C}(Q_1 \times Q_2)$

A	**B**	**D**
1	2	1
5	10	6

$\sigma_{A=D}(Q_1 \times Q_2)$

A	**B**	**C**
1	2	1
2	4	2
3	6	10
5	10	6

$\sigma_{B=C}(Q_1 \times Q_2)$

A	**B**	**D**
2	1	1

$\sigma_{B=D}(Q_1 \times Q_2)$

A	**B**	**C**
1	3	1
3	6	20

The remaining consolidations are all intersections of the minimal consolidations, such as:

$$\sigma_{A=C=D}(Q_1 \times Q_2), \sigma_{A=C \wedge B=D}(Q_1 \times Q_2), \sigma_{A=B=C}(Q_1 \times Q_2), \ldots$$

and so forth, for a total of ten distinct solutions to the consolidation problem in this example.

$\square$

Given the worst case, one can calculate the worst case size of the solution space, i.e., the number of candidate consolidations.

Theorem 3.1. Given N queries with M attributes each, the size of the consolidation solution space is bounded by $\Omega(b^{(M^2N^2)})$, where is b the number of possible binary comparisons that can be done among individual attributes (e.g., $=, \leq, \geq$ &c.).

Proof. Let $H = \{\mathbf{V}, \mathbf{E}\}$ be a hypergraph. Also, let there be a set of colors $\mathbf{C} = \{c_0, \ldots c_{b+1}\}$. Each hyperedge has a color defined by the function $c(e) : \mathbf{E} \rightarrow \mathbf{C}$. The set of vertices $\mathbf{V}$ is defined as:

$$\mathbf{V} = \{a_{i,j} : i = 1 \ldots N; j = 1 \ldots M\}$$

[5]The definition of a minimal consolidation will be formalized later in this section.

Thus, there is one vertex in the hypergraph for each attribute, i.e., $|\mathbf{V}| = NM$. The set of hyperedges $\mathbf{E}$ is the union of two disjoint subsets:

$$\mathbf{E} = \widehat{\mathbf{E}} \cup \mathbf{E}^{'}$$

The original relations are defined as hyperedges in $\widehat{\mathbf{E}}$ with color c_0:

$$\begin{aligned} \widehat{\mathbf{E}} \quad &= \quad \{\{a_{i,j} : j = 1 \ldots M\} : i = 1 \ldots N\} \\ e \in \widehat{\mathbf{E}} \quad &\Longleftrightarrow \quad c(e) = c_0 \end{aligned}$$

Therefore there are $|\widehat{\mathbf{E}}| = N$ hyperedges with color c_0.

Let the remaining hyperedges (i.e., the set $\mathbf{E}^{'}$) represent binary comparisons between attributes of different relations and be colored in any possible permutation, each color defining the operation to be performed between the two attributes or lack thereof (e.g., c_1 being a non-link (i.e., null operation), c_2 representing an equi-join, c_3 representing a "$\geq$" comparison &c.). Since there are b colors, one for each possible operation, and a color for null operations (c_1) there are a total of $b + 1$ colors, excluding c_0.

Therefore, the set of hyperedges defining these comparisons is:

$$\begin{aligned} \mathbf{E}^{'} \quad &= \quad \{\{a_{i,j}, a_{k,l}\} : i, k = 1 \ldots N; j, k = 1 \ldots M; i \neq k\} \\ e \in \mathbf{E}^{'} \quad &\Longleftrightarrow \quad c(e) = c_i : 1 \leq i \leq (b + 1) \end{aligned}$$

Through standard combinatorics, the number of different binary links between any two attributes, provided they are not in the same relation, is:

$$|\mathbf{E}^{'}| = \binom{|\mathbf{V}|}{2} - |\widehat{\mathbf{E}}| \binom{M}{2}$$

Since the hyperedges in set $\mathbf{E}^{'}$ can each be colored $b+1$ different ways the hypergraph has:

$$|T_{C_G}| = (b+1)^{|\mathbf{E}^{'}|} = (b+1)^{(\frac{1}{2}M^2N(N-1))}$$

colorings. Each coloring represents one possible configuration of the selection expression σ_{C_G} as given in Eqn. 3.4.

The number of candidates occurring from the number of possible projections can be calculated directly from the total number of attributes (i.e., $|\mathbf{V}|$). The number of possible projections is simply the size of the power set given the set of all attributes in all the N relations. Therefore,

$$|T_{\mathbf{P}_G}| = |2^{\mathbf{V}}| = 2^{|\mathbf{V}|} = 2^{MN}$$

The number of candidate consolidations is the product of the sizes of the configuration spaces of $\mathbf{P}_G$ and C_G. Therefore the size of the solution space for consolidation becomes:

$$|T_{\mathbf{P}_G}||T_{C_G}| = 2^{MN}(b+1)^{(\frac{1}{2}M^2N(N-1))}$$

Given that there is at least one operation (i.e., $b \geq 1$), it is evident that the growth rate of the $\mathbf{P}_G$ configuration space is always dominated by the growth rate of the C_G configuration space. Hence, one arrives at the asymptotic bound of $\Omega(b^{(M^2N^2)})$.

$\square$

3.5.1 Minimal Consolidations

Obviously, the enumeration, ranking and refinement of such a large set of solutions is unrealistic except for very small problems. I therefore define a subset of the solution space in which effective query consolidation is possible realistically.

Definition 3.6 (Minimal Consolidation)**.** A minimal consolidation of a set of queries $Q_1 \ldots Q_i$ is a consolidation where the hypergraph representing the consolidation is minimally connected.

In other words, a minimal consolidation is one where the removal of a single clause from the selection expression C_G would cause one of the component queries Q_i to be disconnected from the rest. This would therefore cause that component to become "unnecessary" in terms of the necessity assumption introduced in Sec. 3.4. In that regard, minimal consolidations are the simplest solutions that still satisfy the necessity assumption.

Corollary 3.1. Given N queries with M attributes each, the size of the set of minimal consolidations is bounded by $\Omega(M^2 N^N b^N)$, where is b the number of possible binary comparisons.

Proof. Using the hypergraph in the proof of Thm. 3.1, by definition, minimal consolidations represent spanning trees over the hypergraph such that the vertex set is identical and the hyperedge set is the union of two disjoint sets:

$$\mathbf{E} = \widehat{\mathbf{E}} \cup \mathbf{E}''$$

$\widehat{\mathbf{E}}$ is defined as in the proof of Thm. 3.1, representing the hyperedges defining the base relations. $\mathbf{E}''$ defines the smallest set of links that connect these relations. It is clear from basic graph theory that all configurations of $\mathbf{E}''$ consist of exactly $|\widehat{\mathbf{E}}| - 1$ elements, since there will be one link and one link only connecting each relation to the other.

Using Cayley's formula, if each relation consisted of a single vertex (i.e., a single attribute), there would be $|\widehat{\mathbf{E}}|^{(|\widehat{\mathbf{E}}|-2)}$ possible sets that could link these together. Instead, each relation is a hyperedge of M attributes (vertices). Note that every subgraph of the original hypergraph induced by a pair of edges in $\widehat{\mathbf{E}}$ is a Moore Graph (i.e., complete bipartite graph of the form $K_{M,M}$). Therefore, by definition, there are M^2 possible ways to link each pair of relations.

Hence, the number of possible configurations for the edge set $\mathbf{E}''$ is the product of the number of ways the relations can be paired up and the number of ways each pair can be linked:

$$|T_{\mathbf{E}''}| = M^2 \widehat{\mathbf{E}}^{(\widehat{\mathbf{E}}-2)}$$

Working in a manner similar to the proof of Thm. 3.1, given b possible comparison operations, there are $c_1 \dots c_b$ ways each link may be colored. The final number of colorings is thus:

$$|T_{\mathbf{E}''}| b^{(|\widehat{\mathbf{E}}|-1)} = M^2 N^{(N-2)} b^{(N-1)}$$

leading to the asymptotic upper bound of $\Omega(M^2 N^N b^N)$. $\square$

Therefore, considering only minimal consolidations gives a rich set of solutions while alleviating some the combinatorial explosion inherent in the complete solution set.

Figure 3.2 shows the growths of the two with respect to the number of component views and the number of attributes per view. In both cases the number of binary operations was limited to $b = 1$. This shows the case where there are, for example, only equi-joins between the components. The number of minimal consolidations (Figure 3.2(b)) are the ways of assembling the components that have the minimum number of joins in order to connect all. The number of total consolidations (Figure 3.2(a)) are all the possible ways these components can be joined, including equi-joins over multiple attributes and cyclical joins.

In the subsequent chapter, I will approach the problem initially in terms of evaluating the minimal consolidations as to their "likelihood". This likelihood emphasizes the pairwise join consistency over constituent relations. Since minimal consolidations are by definition spanning trees, they are also acyclic. It has been previously proven [15] that in acyclic schemes local (pairwise) join consistency leads to global consistency. Therefore, if a minimal consolidation is made up of pairwise consistent joins, one can say it will be consistent

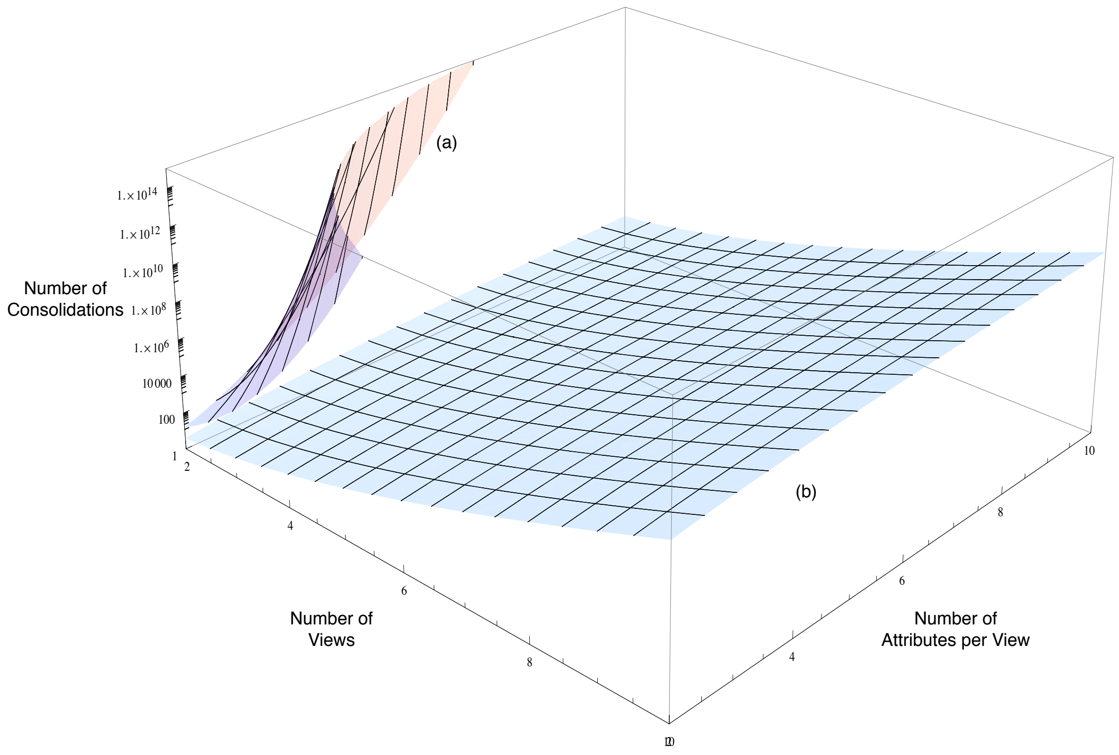


Figure 3.2: **Extent of the Number of Possible Consolidations**
(a) Growth of total number of consolidations ($b = 1$).
(b) Growth of number of minimal consolidations ($b = 1$).
Note: Clearly, the number of consolidations is defined only when the number of views and the number of attributes per view is a positive integer, and is itself always an integer. Nevertheless, the growth functions are continuous over real numbers and hence the surfaces on this plot are thus shown continuously for illustration purposes.

globally. Further on in Chp 6, these minimal consolidations are going to be enhanced by adding new selection operations.

3.6 Summary

I have given definitions for the important concepts underlying information integration, query decomposition, and its reverse; query consolidation. Additionally, I have described the data model and the structure of the global schema used to provide a setting for the remaining chapters. Furthermore, I've introduced the basic assumptions on query language and querying agent behavior.

Finally, the multiplicity of the possible solutions was investigated in this chapter, discussing the reasons for the multiplicity, the extent thereof and the growth of the number of solutions with respect to number of component queries, their respective arities and the number of comparison operations possible. A new subclass of solutions termed *minimal* consolidations was also introduced and analyzed. This subclass will form the basis of the solution method in the next chapter.

Chapter 4: Evaluating Minimal Consolidations and Join Inference

In the previous chapter, the number of consolidations possible over a given number of views with a given number of common attributes was discussed. The results of that analysis is, at first, discouraging. However, with several assumptions and heuristics, most of the solutions can and will be pruned, with the remaining solutions being ranked in order of likelihood.

The number of possible consolidations being impractically large, I will focus instead on the minimal consolidations. The reasons for this are threefold. First of all, as seen previously, the number of minimal consolidations for a given problem are orders of magnitude lower than that of all possible consolidations.

Secondly, minimal consolidations are represented by acyclic graphs, which makes it possible to enumerate, evaluate and rank them easily. This comes from the fact that in an acyclic organization, local evaluation of certain aspects (such as pairwise lossless-ness of joins) are distributive to the global level. If the join plan is acyclic, pairwise evaluation of join strengths (i.e., degrees of the join dependency) can be summed up to obtain the same for the whole plan.

Finally, all possible consolidations are derivable from at least one minimal consolidation. This means that for any possible solution of the form:

$$Q = \Pi_{\mathbf{P}_G} \sigma_{C_G} (V_1 \times \ldots \times V_n)$$

for a given consolidation problem, there exists a minimal consolidation Q' of the same problem and a selection expression C'_G such that:

$$Q = \sigma_{C'_G}(Q')$$

In other words, minimal consolidations can be used as precursors to more complex consolidations by the introduction of additional selection constraints. Furthermore, I limit the number of comparisons possible to one[1], namely equality testing. Given this limitation, it clear that the minimal consolidations of this form are *join plans*. Specifically, since all minimal consolidations are trees, the join plans considered are star and chain joins and hybrids thereof.

I will start by discussing the construction of a join graph from which the join plans are derived. From there I will discuss how the Multiplex global scheme is used to select and rank those minimal consolidations that make sense in the real world.

In the second part of the chapter, I will consider the case where the global schema cannot be used. This happens when one or more of the views that are to be consolidated cannot be mapped onto the global schema, or they do not have any dependency links to the other views and thus have to be analyzed extensionally in order to discover these links.

4.1 Methodology

In the previous chapter, I discussed the possible number of minimal consolidations from a completely combinatorial aspect. In actuality, a large number of the minimal consolidations do not make sense semantically. Consider the following example:

Example 4.1. Consider the following two relations retrieved from two different sources (e.g., a list of employees of ABC Corp. and the list of accounts at XYZ Bank) :

Employees:

EmpID	**Name**	**Phone**
222-99-3034	Alice	555-3456
342-23-3244	Bob	555-3422
234-43-4322	Carol	555-3422

Accounts:

SSN	**Tel**	**Balance**
222-99-3034	555-3456	\$5644.03
342-23-3244	555-3422	\$56.34
234-43-4322	555-3422	\$3456.08

[1]i.e., $b = 1$ using the nomenclature of Chp. 3.

Immediate inspection by a human, even without any metadata, reveals that there are two ways of joining these tables, namely joining with SSN = EmpID or Tel = Phone and one can see right away that the remaining seven combinations of these two relations will not succeed. Upon closer inspection it is apparent that SSN = EmpID is a better way to join these two tables since Bob and Carol have the same telephone number (meaning they share the same house/office line). □

The monitor can use its own global schema (defined in section 3.2) to prune meaningless joins and therefore arrive at a subset of the minimal consolidation solutions that actually make sense in the real world. Recall that the monitor's global schema contains a set of generalized dependencies $\mathbf{\Sigma}$. These dependencies will indicate which joins between an arbitrary set of answer fragments are meaningful in constructing a global answer.

The proposed monitor will contain mappings between a global schema and the local sources that have been mapped to this schema. This mapping architecture is equivalent to an information integration system, which is Multiplex.

In accordance with Multiplex, the mappings are to follow the GLAV paradigm, in which views of the local sources are mapped into views of the global schema. Figure 4.1 illustrates the idea. Note that if the local view for a mapping covers the entire source, the mapping effectively becomes a LAV mapping for that source. Likewise, if the global view of the mapping covers the whole global schema the mapping is a GAV mapping. It is therefore possible to achieve generality in the mappings.

Once a global schema that unifies the local sources is available, any queries that are posed to a local source can be expressed as queries acting on the global view of the source. This gives one the ability to manipulate and relate queries on widely different sources on a common substrate (i.e., the global schema). Therefore, while monitoring queries sent to different sources, the system will map these onto the global schema and can thus consolidate them as a single global query.

In rough strokes, the overall approach may be sketched as follows. I assume a virtual database is available that integrates local databases $\mathcal{L}_1, \ldots, \mathcal{L}_n$ in a global scheme $\mathcal{G}$. Given

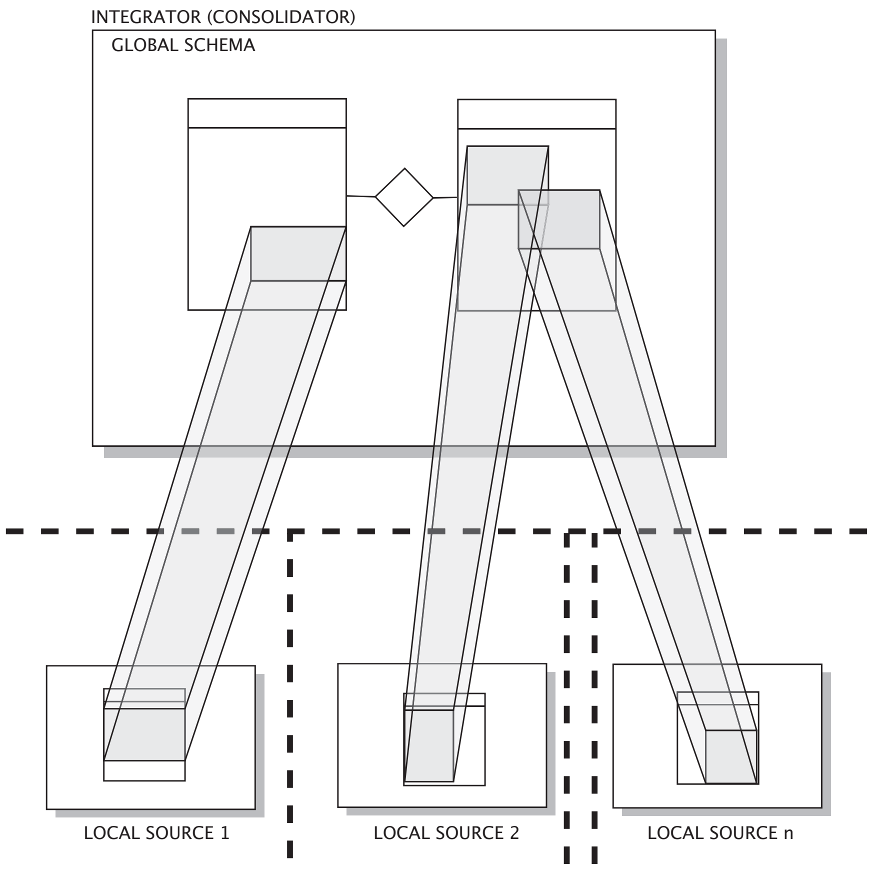


Figure 4.1: **Mapping of Local Sources to the Global Schema**

local queries $Q_1, \ldots, Q_n$, I start with a procedure that roughly reverses query decomposition:

1. For each local query Q_i, determine the views U_j that are relevant (intersect with Q_i).
2. Materialize the extensions $\mathbf{Q}_i$ to obtain the part $\bar{\mathbf{Q}}_i$ that is within U_j.
3. In the virtual database, materialize the corresponding views V_j with the answers $\mathbf{Q}_i$.
4. Populate the base relations R_k with materialized views V_j.

As described in steps 1 and 2, it is possible that a local query Q_i would not be contained in a local view U_j, causing some data to be discarded when global structures are populated. As this will decrease the effectiveness of the consolidation, I assume that all local queries are contained in mapped views. I will relax this assumption in Sec. 4.2, later on.

Upon materializing the views V_j from the received answers $\mathbf{Q}_i$, and then populating the relations R_k with these views, one finds that a view may be contained in a relation, or several views may be contained in the same relation, or a view may span several relations. Consider the example in Figure 4.2: V_1 spans relations R_1 and R_2, both V_2 and V_3 are contained in R_3, and V_4 is contained in R_4. The task now is to join the relations that received data; all other relations are ignored.

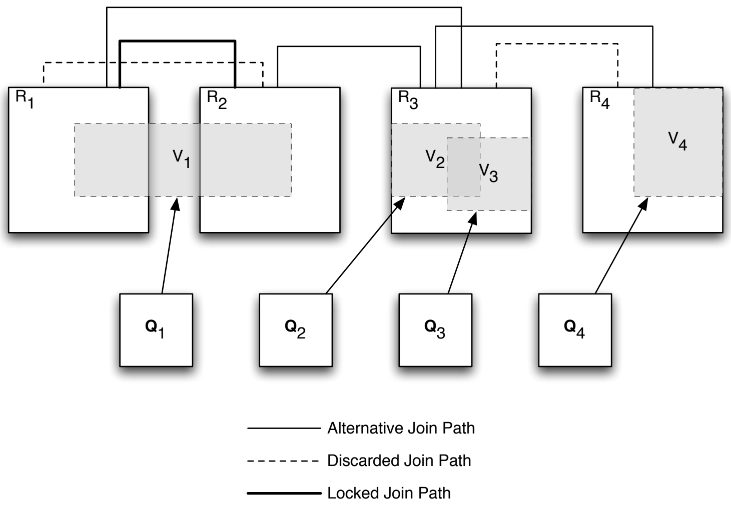


Figure 4.2: **View Mapping and Join Paths**

The global scheme contains dependency and dictionary information (essentially, foreign key constraints) that establishes possible relationships among its relations. Figure 4.2 also shows the relationships among the four relations. Initially, I ignore the relationships that cannot be used because none of their attributes were materialized. These are shown as dashed lines. Furthermore, any relations that are spanned by a single view are considered to be joined unambiguously. Therefore, the join implied by the spanning view is "locked" and all its alternatives (i.e., other join paths between the two relations) are ignored. Locked joins are shown in bold lines. Thus, I obtain a multigraph in which vertices are relations and edges are possible joins. The join graph for the example is given in Figure 4.3.

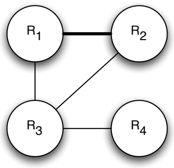


Figure 4.3: **Join Graph of the Mapping in Fig. 4.2**

A join plan is a tree that spans this graph. Such a join plan is also, by definition, a minimal consolidation. One can therefore obtain all the possible minimal consolidations (join plans) by enumerating the spanning trees in the join graph. Mandatory edges indicating locked joins (such as the one between R_1 and R_2) are handled by merging their two end vertices.

4.2 Join Inference without Global Schema Information

In this section I consider the problem of inferring joins among relational (tabular) data instances in the absence of a comprehensive schema that describes the data. In other words, given two or more pieces of tabular information the problem is of deciding if, and how, these tables may be joined in a way that the resulting table contains the relevant tuples of the input tables such that they are related.

In a way the method proposed here acts as a simplified ad-hoc relational information integration system. Given a set of tables with no information about the columns and of how they are related to each other it attempts to find the most likely universal relation [64] with the given pieces.

Apart from query consolidation, a schema agnostic join inference method has appeal as a data source discovery and assimilation tool for unstructured or semi-structured sources [19], the most prominent of which is the World Wide Web. The sources on the Web that are exploitable by information integration systems currently are those with well defined access methods and rich metadata (e.g., XML based web services). There is however, a large amount of information on the web which is published as tables with no such descriptive information. These include text documents with embedded tables, web pages with HTML tables and even older text which has been scanned with OCR systems and put online. All of these documents have tabular (i.e., relational) data that can be satisfactorily extracted from text automatically [78]. Even then, however, these are semi-structured data with no descriptions as to their meaning in a global schema or their dependencies. An automated join inference method, if efficient, can analyze a large number of such pieces rapidly and find any possible join links between these and the extensions of existing parts of a virtual database, thereby allowing them to be recruited to further increase the completeness of the information. Alternatively, the method proposed here can also be used as a data mining tool in its own right in collecting pieces of relational data strewn over multiple documents and joining them into larger, global relations that can then serve as fact tables or existing dimensions to fact tables in a data warehouse and can be mined for knowledge.

The first step in finding possible joins among unmarked relations is identifying the attributes that are compatible in their domains. There are various methods[2] for matching attribute domains in previous literature. I will introduce a simpler method that is able to work with scarce information and in polynomial time by exploiting certain aspects of key attributes. Even when matching by domain is done, there may be multiple join paths

[2]See Chp. 2

available due to more than one pair of attributes matching. In order to solve this ambiguity, I make the assumption that *likely joins paths are those that employ key-foreign key relationships among the relations*. While it can be argued that there may be cases where interesting joins may be obtained by joining on non-key relationships I will limit the scope to join plans that are lossless or, if there are no such options, to join plans that approximate the same as much as possible.

The terms in the conjunct are the equi-join links between pairs of relations. One thus needs to find the set of links which are foreign key relationships such that all relations are connected in a single tree. If every input table is guaranteed to have at least one such good link to another table, then one can find a minimal consolidation. However, given that the data coming from unstructured and independent sources is liable to be flawed both intensionally and extensionally, such perfect linkage is not guaranteed by any means. Thus one must be able to find the closest match, in other words the maximally contained consolidation with the available data. I approach this problem using an information theoretic measure by which I score each possible link between two relations with a value defining to what extent that link constitutes a foreign key relationship. Thereafter, the tree of links spanning all the input relations and having highest total score is the maximally contained one. Furthermore, one is able to rank alternative trees (i.e., join plans) by their total scores, thus achieving a method that has better recall.

4.2.1 The Join Graph

I model the set of input queries with an undirected and labeled multigraph $G = (\mathbf{V}, \mathbf{E})$, termed the *join graph*. The vertices $\mathbf{V}$ are the input relations $R_1, \ldots, R_n$, and the edges $\mathbf{E}$ are possible equi-joins between the corresponding relations. As two relations could be joined on different pairs of attributes, there may be parallel edges and the edges are labeled with the join attributes. For example, an edge labeled (A_1, A_2) represents the equi-join $R_1 \bowtie_{A_1=A_2} R_2$.

Figure 4.4 shows an example join graph involving three relations $R_1 = \{A, B, C\}$, $R_2 = \{K, L\}$ and $R_3 = \{X\}$ and a complete set of edges, representing a situation where every pair of attributes from two different relations is a possible equi-join.

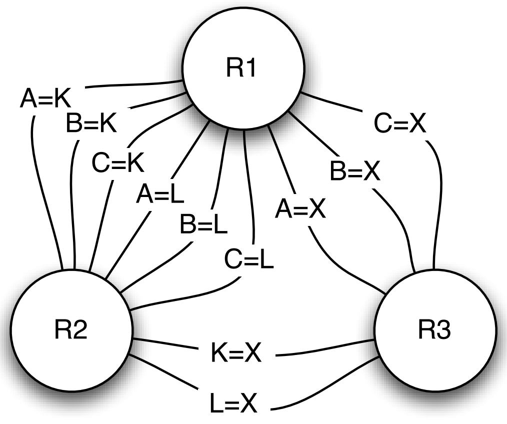


Figure 4.4: **Example Join Graph**

A *join plan* for the given relations is a relational algebra expression in which every relation appears exactly once, and relations are connected with single-attribute equi-joins. In the example of Figure 4.4, $R_1 \bowtie_{A=K} R_2 \bowtie_{L=X} R_3$ and $R_1 \bowtie_{B=X} R_3 \bowtie_{K=X} R_2$ are two join plans. Given a connected join graph, a join plan is represented as a spanning tree.

The goal is not merely to find a possible join plan or to enumerate all the possible join plans. Rather, the goal is to sort the possible join plans according to their *likelihood*. Hence, I describe a method by which each join plan is assigned a *likelihood score* (i.e., a *weight*).

As mentioned previously, because they are acyclical, the join plans are locally optimizable. That is, if given more than two relations, the most likely join plan is one that comprises the most likely individual joins. Consequently, I describe a method for assigning weights to individual joins.

In the graph model, I score each edge in the multigraph with a weight that represents the likelihood of the join it represents. The weight of a join plan is then the total weights of the participating edges, and possible join plans (minimal consolidations) are sorted by their total weight. The most likely join plan is the one that corresponds to a *maximum weight* *spanning tree.*

Example 4.2. Consider Figure 4.5, which shows a weighted version of the graph in Figure 4.4. For simplicity, only 5 of the 11 possible joins are considered. This simple join graph has a total of 8 possible spanning trees. Of these, the maximum spanning tree (shown in bold edges) denotes the join plan $R_1 \bowtie_{B=K} R_2 \bowtie_{L=X} R_3$; The tree in second place denotes the join plan $R_2 \bowtie_{B=K} R_1 \bowtie_{A=X} R_3$; and so forth.

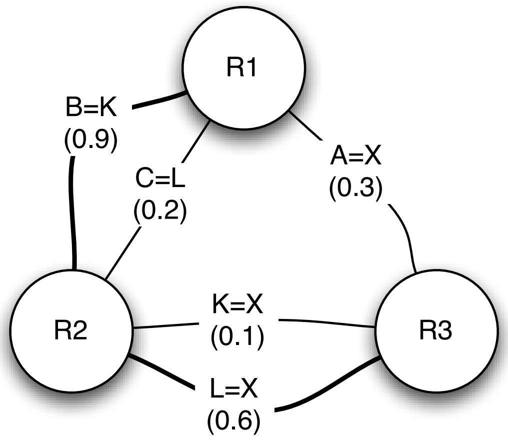


Figure 4.5: **Weighted Join Graph**

Enumeration of spanning trees can be done by a variety of algorithms, and I use the algorithm described in [46], which allows the enumeration of an arbitrary number of spanning trees in order of total weight and can handle multigraphs. Its complexity depends on the number of trees generated and the worst case complexity is $O(N \log V + VE)$, where V is the number of vertices, E is the number of edges and N is the desired number of top-ranked trees to be generated.

4.2.2 Pruning Edges

I now turn to the central issue of assigning weights to equi-joins. Given attribute A_1 of relation R_1 and attribute A_2 of relation R_2, one considers the join $R_1 \bowtie_{A_1=A_2} R_2$ to be optimal (most likely), if A_1 and A_2 have identical domains, and are primary keys of their respective relations. The lesser the degree of satisfaction of these two conditions, the farther the join is from optimum.

To justify, consider the situations at the other extremes. (1) When the domains are completely disjoint, the join is empty; and (2) when the attributes are "anti-keys" (i.e., all tuples of R_1 have the same value in their A_1 attribute, and all the tuples of R_2 have the same value in their A_2 attribute), the join is either empty (when these values are different), or the entire Cartesian product of R_1 or R_2 (when they are identical). In either of these extreme cases, the join $R_1 \bowtie_{A_1=A_2} R_2$ is farthest from optimal.

The optimal join can be viewed as a foreign-key join, in which every tuple in one relation matches exactly one tuple in the other relation (a 1:1 matching). When the domains are not identical, there will be tuples in each relation that are not matched. And when the attributes are not strict keys, tuples in one relation may match several tuples in the other relation.

Hence, the likelihood of a join $R_1 \bowtie_{A_1=A_2} R_2$ is derived from these two parameters:

- The *compatibility* of $dom(A_1)$ and $dom(A_2)$.
- The *selectivity* of A_1 on R_1, and the selectivity of A_2 on R_2.

In a preliminary step, I identify those attributes that are considered improbable for joins. These include attributes that contain long character strings or real numbers. These attributes are discarded.[3] Next, for the remaining attributes one can construct a compatibility matrix. Attribute compatibility is calculated using this domain similarity measure:

$$sim(A_1, A_2) = \frac{|dom(A_1) \cap dom(A_2)|}{|dom(A_1) \cup dom(A_2)|} \tag{4.1}$$

The values of this measure are between 0 and 1. The measure is 0 when the attribute domains are disjoint, and 1 when they are identical. Using a threshold, the compatibility

[3]In the experiments (to be detailed in Chp. 8) I consider attributes with strings exceeding 10 characters to be unlikely participants in joins. Clearly, the definition of attributes that are join candidates can be relaxed.

matrix is binarized. The join graph is then constructed with an edge for every two attributes judged compatible.

There is an anomaly that may occur in some join graphs that is corrected at this stage. As an example, consider a relation *Country* that lists countries, and a relation *Border* that lists pairs of countries that share a border. In essence, *Borders* models a many-to-many relationship between entities of *Country.* Unfortunately, such occurrences lead to cyclical join graphs and must be remedied. Figure 4.6(a) shows the initial join graph. Obviously, the desirable join plan is *Country* $\bowtie$ *Borders* $\bowtie$ *Country*, yet the spanning tree approach will not discover this join plan. This anomaly is rectified if the join graph includes two instances of *Country*, each with its own edge to *Borders*, as shown in Figure 4.6(b). Such situations are discovered by searching for edges from two different attributes of one relation to the same attribute of another relation. When such an occurrence is found, the vertex of the latter relation is split in two, with each vertex receiving one of the edges.

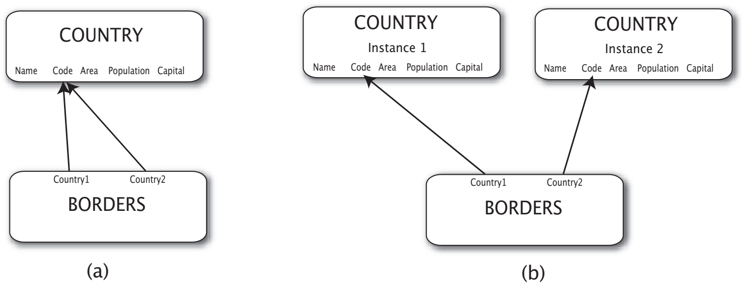

COUNTRY
Name Code Area Population Capital
Country1 Country2
BORDERS
(a)
COUNTRY
Instance 1
Name Code Area Population Capital
COUNTRY
Instance 2
Name Code Area Population Capital
Country1 Country2
BORDERS
(b)

Figure 4.6: **Resolution of Join Cycles**
(a) Unresolved join cycle
(b) Reinstantiation to resolve cycle

4.2.3 Weighting Edges

Having constructed a join graph based on attribute compatibility, I now describe how to score its edges on the basis of attribute selectivity. As mentioned earlier, the method quantifies the levels to which attribute relationships obey referential constraints. The method is based on the concept of *conditional entropy* in information theory.

The entropy of an attribute A is defined as

$$H(A) = \sum_{v \in dom(A)} -p(A = v) \cdot \log_2 p(A = v)$$

where $p(A = v)$ is the proportion of tuples in which the value v occurs. Intuitively, $H(A)$ measures the uniformity of the distribution of the values of A. Assuming n distinct values, entropy ranges between 0 and $\log_2(n)$. The maximal value corresponds to perfect uniformity of distribution (lowest information content). For example, when $dom(A)$ contains 4 distinct values, and each occurs 5 times, then $H(A) = \log_2(4) = 2$. In this case, $H(A)$ is also the number of bits required to represent the values of $dom(A)$. The entropy of a relation R is thus the sum of the entropies of its attributes, i.e.,

$$H(R) = \sum_i H(A_i) \tag{4.2}$$

Let A be an attribute of R, and consider a "horizontal slice" of R, created by requiring A to have a specific value v. Denote the entropy of this slice $H(\sigma_{A=v}(R))$. As $dom(A)$ is the set of all the values of attribute A, its values partition R to several such slices, each with its own entropy. The average of these slice entropies weighted by the prior probability of the individual values is the expected value of the relation given a value for attribute A. This is termed the *conditional entropy* of R given the attribute A:

$$H(R|A) = \sum_{v \in dom(A)} p(A = v) \cdot H(\sigma_{A=v}(R)) \tag{4.3}$$

This conditional entropy reflects a reduced level of uncertainty when R has been partitioned by its A values. The difference between the prior and posterior entropies quantifies this reduction. For a meaningful comparison with reductions obtained in other relations using other attributes, I normalize this difference by dividing it by the prior relation entropy.

The result is the *information gain*[4] in R given attribute A:

$$I(R|A) = \frac{H(R) - H(R|A)}{H(R)} \tag{4.4}$$

Assume A_1 is attribute of relation R_1 and A_2 is attribute of relation R_2, and consider the join $R_1 \bowtie_{A_1=A_2} R_2$. For each tuple of R_1 the value of A_1 is compared to the value of A_2 in each tuple of R_2. $dom(A_1) \cap dom(A_2)$ is then the set of values of A_1 that find a match in R_2. Therefore, in the aftermath of the join, the relation A_2 is partitioned by the values in the set $dom(A_1) \cap dom(A_2)$.

The information gain of R_2 in the aftermath of this join is therefore

$$I(R_2|A_2) = \frac{H(R_2) - H(R_2|A_2)}{H(R_2)} \tag{4.5}$$

Note that the posterior entropy $H(R_2|A_2)$ is calculated as the expected values of entropies for slices created by values in $dom(A_1) \cap dom(A_2)$.

If A_2 is a key of R_2, then each of the slices has a single tuple. Therefore each of the slice attributes contains just one value and its entropy is 0. Since slice entropy is the sum of its attribute entropies, each slice entropy is 0. Consequently, the posterior entropy of R_2 is 0, as well, and the information gain of R_2 in the aftermath of the join is 1.

If A_2 is anti-key ($dom(A_2)$ has but a single value), then a single slice is created, the posterior entropy of R_2 is the same as its prior entropy, and the information gain in the aftermath of the join is 0.

In between these extreme cases, the information gain is a value between 0 and 1. The information gain will be higher when slice sizes are more uniform, and the tuples in each slice are more homogeneous. In other words, the more an attribute acts like a key, the closer its gain will be to 1.

[4]Alternatively called the *KullbackLeibler Divergence* [52]

This method may be regarded as an information theoretic way of quantifying the strength of referential constraints. If a foreign attribute produces an information gain of 1 in another relation, that relation is functionally dependent on the attribute. Indeed, this approach generalizes the definition of dependency from a binary concept to a gradual one.

Note, however, that the join operation is symmetrical, and thus induces a dual information gain in R_1:

$$I(R_1|A_1) = \frac{H(R_1) - H(R_1|A_1)}{H(R_1)}$$

The weight assigned to the edge that denotes the join $R_1 \bowtie_{A_1=A_2} R_2$ is therefore a pair of values, expressing the information gain of each of the join participants:

$$w(A_1, A_2) = (I(R_1|A_1), I(R_2|A_2)) \tag{4.6}$$

Example 4.3. Consider the relations

$$\begin{aligned} R(A,B) &= \{(1,b_1),(2,b_2),(3,b_3),(4,b_4)\} \\ S(A,C) &= \{(2,c_1),(3,c_2),(4,c_3),(5,c_4)\} \end{aligned}$$

and a join between $R.A$ and $S.A$. It is a one-to-one matching and it results in three tuples. Consider now the effect of the join on the entropy of S. Initially, the entropy of the attributes of S are $H(S.A) = 2$ and $H(S.C) = 2$, and the apriori entropy is therefore $H(S) = H(S.A) + H(S.B) = 4$. When the join iterates over the four values of $R.A$ it creates in S four slices: $A = 1$ creates an empty slice, $A = 2$ creates $\{(2, c_1)\}$, $A = 3$ creates $\{(3, c_2)\}$, and $A = 4$ creates $\{(4, c_3)\}$. Each slice has entropy $0 + 0 = 0$, and the posterior entropy is therefore $H_A(S) = (0 + 0 + 0 + 0)/4 = 0$. Consequently, the information gain for S from this join is $I_A(S) = (4 - 0)/4 = 1$. In this case, the information gain for R from this join would be identical: $I_A(R) = 1$.

As a second example, consider

$$\begin{aligned} R(A,B) &= \{(1,b_1),(2,b_2),(3,b_3),(4,b_4)\} \\ S(A,C) &= \{(2,c_1),(2,c_2),(3,c_3),(3,c_4), \\ &\quad (4,c_5),(4,c_6),(5,c_7),(5,c_8)\} \end{aligned}$$

and a join between $R.A$ and $S.A$. It is a one-to-many matching in which every tuple of R matches zero or two tuples of S, and it results in six tuples. The apriori entropy of S is $H(S) = H(S.A)+H(S.B) = 5$. When the join iterates over the four values of $R.A$ it creates in S one empty slice and three slices with two tuples each: $\{(2,c_1),(2,c_2)\}$, $\{(3,c_3),(3,c_4)\}$ and $\{(4,c_5),(4,c_6)\}$. The empty slice has entropy 0, and each of the other three slices has entropy $0+1=1$, and the posterior entropy is therefore $H_A(S) = (0+1+1+1)/4 = 0.75$. Consequently, the information gain for S from this join is $I_A(S) = (5-0.75)/5 = 0.85$. The information gain for R from this join would be $I_A(R) = 1$.

Next, consider a join between

$$\begin{aligned} R(A,B) &= \{(1,b_1),(2,b_2),(3,b_3),(4,b_4)\} \\ S(A,C) &= \{(2,c_1),(2,c_2),(3,c_3),(4,c_4),(4,c_5),(5,c_6)\} \end{aligned}$$

It is a one-to-many matching in which tuples of R match different numbers of tuples of S, and it results in 5 tuples. The apriori entropy is $H(S) = 4.5$ and posterior entropy is $H_A(S) = 0.66$. The information gain for S is $I_A(S) = 0.85$. The information gain for R would be $I_A(R) = 1$.

Next, consider

$$\begin{aligned} R(A,B) &= \{(1,b_1),(1,b_2),(2,b_3),(2,b_4), \\ &\quad (3,b_5),(3,b_6),(4,b_7),(4,b_8)\} \\ S(A,C) &= \{(2,c_1),(2,c_2),(3,c_3),(3,c_4), \\ &\quad (4,c_5),(4,c_6),(5,c_7),(5,c_8)\} \end{aligned}$$

It is a many-to-many matching in which every tuple of R relation matches zero or two tuples of S. Both information gains are identical: $I_A(S) = 0.85$ and $I_A(R) = 0.85$.

Finally, consider

$$\begin{aligned} R(A,B) &= \{(1,b_1),(2,b_2),(2,b_3),(2,b_4), \\ &\quad (3,b_5),(4,b_6),(4,b_7),(4,b_8)\} \\ S(A,C) &= \{(2,c_1),(2,c_2),(3,c_3),(3,c_4), \\ &\quad (4,c_5),(4,c_6),(4,c_7),(5,c_8)\} \end{aligned}$$

It is a many-to-many matching in which tuples of R match different numbers of tuples of S. The information gains are $I_A(S) = 0.82$ and $I_A(R) = 0.83$. $\square$

The examples demonstrate how the method is sensitive to the selectivity of the join (on both participating relations). A join matching a single tuple is scored with perfection, and as the average number of tuples matched increases, the score decreases. Therefore, on average, a one-to-n join will be scored higher than a one-to-m join when $n < m$.

When ranking spanning trees I use the higher of the two values to decide between two different edges; the second value is used to resolve ties.

Notice that this method of approximating key relationships is superior to methods in previous literature. A commonly used measure is g_3 [49]:

$$g_3(X \to Y, \mathbf{R}) = \frac{|\mathbf{R}| - \max\{|\mathbf{S}| \mid \mathbf{S} \subseteq \mathbf{R}, \mathbf{S} \models X \to Y)\}}{|\mathbf{R}|}$$

In essence the value of g_3 is the fraction of a relation R that satisfies a functional dependency, and therefore a key. While straightforward and logical, this measure neglects the effects of data quality. Especially in distributed heterogenous database setups, intensionally identical tuples may be extensionally inconsistent between two sources.

Example 4.4. Consider the following relations:

R:

ID	**Name**	**Phone**
1	Alice	9345
2	Bob	9342
3	Carol	9341

R′:

ID	**Name**	**Phone**
1	Alice	9345
2	Bob	9345
2	Bob	9343
3	Carol	9341

Here, **R′** is an extensionally inconsistent form of **R**. Notice that Bob is shown twice, with different telephone extensions, possibly due to a union from two sources with different information (e.g., older phone number, or typo). Now compare the values of g_3 and the information gain measures:

X	$g_3(ID \to X)$	$I_{ID}(X)$
R	1	1
R'	0.5	0.93

The discrete (i.e., all or nothing) nature of the g_3 measure neglects the fact that the ID field is very close to being a proper key in **R′**. The information gain measure, however, is able to capture that fact and thus is more tolerant of rare inconsistencies in an attribute, especially if the remainder of a tuple is functionally dependent on the key.

4.2.4 Cost Analysis

To help analyze the cost of the method for assigning weights to the edges of a join graph, Figure 4.7 summarizes the method in algorithm format. In this figure, the set *Candidates* accumulates the edges that pass the compatibility test, and θ is the threshold of compatibility.

```
 1: $Candidates \leftarrow \emptyset$
 2: for all $A_i$ do
 3:    if $A_i$ is a viable join attribute then
 4:       $Candidates \leftarrow A_i$
 5:    end if
 6: end for
 7: for all $A_i \in Candidates$ do
 8:    for all $A_j \in Candidates$ do
 9:       if $i \neq j$ and $A_i$ and $A_j$ are in different relations then
10:          if $sim(A_i, A_j) < \theta$ then
11:             Discard edge $(A_i, A_j)$
12:          else
13:             Calculate $w(A_i, A_j)$
14:          end if
15:       end if
16:    end for
17: end for
```

Figure 4.7: **Pruning and Weighting Algorithm**

Let K denote the total number of relations, P the average arity, and N the average cardinality. Typically, K and P are small, whereas N could be very large. The initial pruning phase (lines 1–6) can be done by a single pass of each relation, and is therefore of complexity $O(KN)$.

The calculation of similarity of two attributes (line 10) requires finding the cardinality of the union and intersections of the two domains. This can be accomplished by sorting the two relations on these attributes and then traversing both relations simultaneously. The time required for calculating a single similarity is therefore of complexity $O(N \log N)$.

The calculation of weights (line 13) is slightly more complicated. Consider the join $R_1 \bowtie_{A_1=A_2} R_2$. First, each relation is sorted on its join attribute. Next, R_1 is traversed, and for each new value of A_1, the matching tuples of R_2 (if any) are written out. These constitute a slice. To calculate the posterior entropy of this slice, it is sorted P times; after each sort, the slice is traversed and the posterior entropy of one attribute is calculated. Eventually, the posterior entropy of the slice is derived by summation. The number of slices is $V = dom(A_1) \cap dom(A_2)$ and each slice has on average N/V tuples. Hence, the total cost is $2N \log N$ for the sorts, $2N$ for the traversals, and $VP((N/V)\log(N/V) + N/V)$ for sorting and traversing each of V slices P times. Hence, the cost of calculating the posterior entropy $H(R_2|A_2)$ is at worst $O(PN \log N)$. Calculating the prior entropy $H(R_2)$ is done by P sorts and traversals, with cost $O(PN \log N)$. Altogether, the calculation of each information gain is of complexity $O(PN \log N)$.

To estimate the cost of the entire algorithm I add the cost the initial pruning and the cost of calculating similarity and weights for every pair of attributes from different relations. At worst, there will be P^2K^2 such pairs.[5] Altogether, the cost of generating the weighted join graph is $O(P^3K^2N \log N)$.

4.3 Optimization of the Join Inference Method

Clearly, the most time consuming steps of the algorithm are the calculations of $sim(A_i, A_j)$ and $w(A_i, A_j)$. This computational task may be reduced by using a Monte Carlo approach. Instead of using the complete relations R_i I use subsets R'_i obtained by random samples. From each sample of a relation, I obtain a sample of each of its attributes by a projection. Random sampling of a relation can be done in $O(N)$ time (a single pass of each relation is required). Assuming the sampling rate $|R'_i|/|R_i|$ is such that the samples can fit into memory, it will be possible to perform these two computations quickly.

However, calculating $sim(A_i, A_j)$ on samples presents problems. For example, assume A_i and A_j are key attributes and $dom(A_i) = dom(A_j)$. Even when sampling as much as half of the domains, one could not guarantee a match. I address this problem by using classifiers to summarize domains. As discussed in Sec. 2.4, several classifiers have been proposed to match domains of attributes. Most of these are general in nature and relatively expensive

[5]Note that each edge requires *two* calculations, one for each of its end attributes.

to train. In this case, however, the domains of attributes that survived the initial pruning step have certain helpful characteristics.

Join attributes are generally either integers or short strings of characters (or a combination of these); for example, ID numbers, telephone numbers, model numbers, geographic locations, &c. Such values are often described by simple regular expressions. I therefore summarize the domains by representing them as small finite automata that recognize the strings in a domain. For example, the domain of US Social Security numbers of the form 123-45-6789 may be summarized as the regular expression $(d^3 - d^2 - d^4 : d \in \{0, 1, 2, \ldots, 9\})$

To accomplish this, I use the ID algorithm proposed by [12] and extended in [76] which gives a canonical deterministic finite automaton (DFA) that will recognize a set of strings. The complexity of the algorithm is $O(NP|\Sigma|)$, where N is the number of states, P is the size of the training set, and $|\Sigma|$ is the size of the alphabet. A simple alphabet that can be used is one that includes one symbol for all letters, one symbol for all digits, and several symbols for punctuation marks. The number of states in the DFA cannot be larger than the size of the alphabet times the longest string in the language. Since the alphabet is small and the strings in the domains are small as well, both $|\Sigma|$ and N are small with respect to P. Thus, one can bound the complexity of creating a DFA to $O(P)$. Recall that P is the size of the training set, namely the size of the sample.

Let A'_1 and A'_2 be samples of $dom(A_1)$ and $dom(A_2)$, respectively. Based on these samples two DFAs are constructed, denoted D_1 and D_2, respectively, to recognize these domains. Given a set of strings S, let $D_i(S)$ denote the proportion of strings recognized by D_i. I now define a new similarity measure to quantify the compatibility of two domain samples A'_1 and A'_2. $D_1(A'_2)$ measures the level to which a DFA trained to recognize A'_1 recognizes the set A'_2. Hence, it can be viewed as measuring the similarity of A'_2 to A'_1. In duality, $D_2(A'_1)$ quantifies the level to which a DFA trained to recognize A'_2 recognizes the set A'_1. Hence, it can be viewed as measuring the similarity of A'_1 to A'_2. These two "directional" similarity measures are multiplied to create the new measure of similarity, denoted sim':

$$sim'(A'_1, A'_2) = D_1(dom(A'_2)) \cdot D_2(dom(A'_1))$$

The new measure, which is calculated on domain samples, will be used to measure the compatibility of the domains.

The other costly calculation is that of information gain (which provide the weights). Here, too, one can define approximate information gain. Equation 4.5 expressed the information gain of R_2 in the aftermath of a join $R_1 \bowtie_{A_1=A_2} R_2$. Instead of using the entire relation R, one could use a sample R' of R, and define approximate information gain:

$$I'(R'_2|A'_2) = \frac{H(R'_2) - H(R'_2|A'_2)}{H(R'_2)}$$

One can then use $I'(R'_2|A'_2)$ to estimate $I(R_2|A_2)$. Because the information is merely a sample of the original, the time required to calculate the estimation is much smaller than the time required to calculate the actual information gain.

Recall, however, that the calculation of $H(R_2|A_2)$ used only the values of $dom(A_2)$ that are present in $dom(A_1)$. Hence, when working with samples, one needs to calculate $dom(A'_1) \cap dom(A'_2)$. But, as discussed earlier in this subsection, approximating the intersection of domains with the intersection of samples is not a viable option.

For a moment, assume that the two domains are identical. Then $dom(A'_1) \cap dom(A'_2) = dom(A'_2)$, and one could use $dom(A'_2)$ (no need to intersect). Since this assumption cannot be made safely, I correct the information gain obtained by multiplying it by the selectivity of the DFA for recognizing $dom(A'_2)$ as it is applied to $dom(A'_1)$ (i.e., I reduce the information gain by the same factor that D_1 would have reduced $dom(A'_2)$). Altogether one obtains this estimator for the weight pair:

$$w'(A_1, A_2) = (I'(R'_1|A'_1) \cdot D_1(A'_2),\ I'(R'_2|A'_2) \cdot D_2(A'_1))$$

The new calculation has two advantages with respect to complexity. First, the entire process is performed on a sample of N (which is often an order of magnitude smaller). Second, and more significant, is the number of times that similarity and weight are calculated. This is due to the fact that the need to intersect domains has been removed from both of these calculations. Previously, a given attribute A_j in relation R_i needed to be compared with each of the P attributes of each of the other relations, essentially PK times. Since the calculations now are essentially independent of the other attributes (for example, the new information gain of $R_i.A_j$ is calculated just once, and only corrected for each counter-attribute $R_k.A_l$ by the acceptance rate of the latter domain by the DFA of the former domain, which is also calculated just once), the new complexity is only $O(P^2KS\log S)$, where S is the sample size. As the experiments described in the next section demonstrate, this reduction is very significant.

4.4 Summary

This chapter explains the enumeration of join plans representing possible minimal consolidations and their ranking according to likelihood. The method begins with the mapping of the answers (i.e., extensions) of the component queries onto a global schema and builds a join graph encompassing all the possible joins among the components. The join plans are spanning trees on this join graph.

The scoring of join plans in terms of likelihood and their ranking is done by analyzing the consistency of the joins between pairs of components and selectivity (i.e., *key-ness*) of the join attributes. The selectivity is scored using an information theoretic approach based on information gain. Two forms of the approach are described: an exact yet computationally expensive variant and a sampling variant that is fast yet less accurate.

Chapter 5: Query Sequence Analysis

5.1 Introduction

Thus far I have discussed query consolidation in terms of a set of queries and their materialized answers. However, in the real world, databases receive queries from a given user as a sequence over time. There is no marker that indicates which queries in a sequence constitute a meaningful set. In the absence of such information, all the consolidating system has are logs of queries for local sources. The set of queries that are part of a global query must therefore be extracted from the sequence in the logs or collated in real time as these queries arrive. Furthermore, it would be very useful to the consolidation effort to individually label the queries in a segment with their probable roles in the bigger picture. The purpose of this chapter is the investigation of this problem.

5.2 Problem Overview and Assumptions

Throughout this chapter, 'goal' will be used in the same sense as 'session'. In previous literature, the term 'session' is used in multiple meanings. Occasionally, it is used to indicate the range of transactions between a login-logout cycle or during a given timeframe. In the scope of the consolidation problem, however, a session is generally meant to denote a single information retrieval task of the user, either done in a single query or by combining multiple queries. In order to avoid this ambiguity the set of queries that make up a task of the user will be referred to as a goal.

I start by assuming that one can identify users by some identification mechanism such as a cookie, through authentication or from their network address &c. Therefore the query

Table 5.1: **Intrinsic Features of Query Q_i**

Name	**Notation**	**Description**
Attributes	$\mathbf{Att}_i$	The set of fields projected by the query.
Dependencies	$\mathcal{F}_i^*$	The functional dependency closure of fields in the query.
Constraints	$\mathbf{C}_i$	The set of constraints associated with the query.
Source	D_i	The unique identifier of the local source the query was sent.
Answer Set	$\mathbf{R}_i$	The set of tuples returned in response to the query.
Timestamp	t_i	The universal time at which the query was posed.

sequence being observed is for a single user at a time. However, the sequence is not necessarily of queries sent to the same data source. This data can be collected in two ways. The subject can be monitored directly, either using the logs of the client she is using directly, or through the logs of a proxy or gateway she uses to access the sources. In this case the provenance of the queries in the sequence is more certain (unless someone else is using the terminal). Alternatively, given a common identifier (such as IP or a cookie) one can monitor the logs of the data sources the subject uses. In this case the queries belonging to the same user are collected from the various data sources and are compiled into a single sequence, in order of time stamps.

In either case the existence of a list of queries in chronological order and coming from the same user can be assumed. The queries will be labelled Q_i in a sequence $\mathbf{Q} = Q_1...Q_n$ such that Q_i is chronologically earlier than Q_{i+1}. For each query Q_i I assume to have a set of *intrinsic features*. These are given in Table 5.2.

Given the methods developed in the previous chapters, these pieces of information are clearly available for every query in the sequence. These intrinsic features will be used to generate more detailed feature functions for the approach of this chapter.

Since the scope is limited to conjunctive queries, each query in the sequence can be thus represented by a Horn clause (or alternatively a relational algebra expression containing only selection, projection and joins).

Example 5.1. Let a query sent to an online bookstore's data service be represented as:

$$Q_{example} = \Pi_{\{ISBN,Title,Year\}}(\sigma_{(AuthorID=\text{``Dickens''})}(\text{Author}) \bowtie_{(Book.AuthorID=Author.ID)} \text{Book})$$

This query has the following intrinsic features:

$$\begin{aligned}
\mathbf{Att}_{example} &= \{ISBN, Title, Year\} \\
\mathcal{F}^{*}_{example} &= \{ISBN \rightarrow Title, ISBN \rightarrow Year, ...\} \\
\mathbf{C}_{example} &= \{(Author.Name = \text{“}Dickens''), (Book.AuthorID = Author.ID)\} \\
D_{example} &= \text{http://soap.mybooks.com/Search.wsdl} \\
\mathbf{R}_{example} &= \{(0679783415, \text{“David Copperfield”}, 1850), ...\} \\
t_{example} &= \text{2008-04-24T20:39Z}
\end{aligned}$$

□

Given a query sequence and the intrinsic features of each member of the sequence, the problem involves two sub-problems:

Segmenting Otherwise known as *boundary detection*, this involves finding when one goal in the sequence ends and the other begins.

Labeling Within each segment (goal) I also seek to find the function of each member query. This aids in reproducing the original goal exactly.

Both problems have analogues in natural language processing. The segmenting problem is reminiscent of *sentence boundary detection* and the labeling is very similar to *part-of-speech tagging*. In order to induce the most useful features for the task, I begin to build a model for the sequence structure.

5.3 Assembly of Goals

In the simplest case, the assembly of a goal, G, is the conjunction of queries Q_i in the sequence. In terms of relational algebra this is essentially the joining of the answer sets,

R_i, into a single universal relation. Even with this simple approach, however, there are complications. It may be the case that the component queries do not offer a join path:

Example 5.2. Consider a sequence of two queries, Q_1 and Q_2 with attributes $\mathbf{Att}_1 = \{SSN, Name\}$ and $\mathbf{Att}_2 = \{Phone, Address\}$. The straightforward conjunction (join) of these will result in a cartesian product, which is all but meaningless. Therefore, perhaps it is better to consider these two queries as two separate goals of one query each, rather than a goal of two queries. However, the arrival of a third query Q_3 with attributes $Att_3 = \{SSN, Phone\}$ would change the picture dramatically. The three queries can now be joined into a cohesive goal of three queries. □

The problem of isolating goals from a sequence of queries can be stated as follows:

Given a sequence of queries $\mathbf{Q} = Q_1 ... Q_n$ coming from a user, find all user goals within the sequence that are *maximally coherent*.

A maximal coherent goal, $\mathbf{G}$, is defined as the largest substring, $\mathbf{G} \sqsubseteq \mathbf{Q}$, of the whole query sequence $\mathbf{Q}$, that is cohesive. The informal definition of coherence is clear from the example above. In order to formalize the defintion further, I define the following feature between any two queries Q_i and Q_j:

$$joinable(i,j) = \begin{cases} 1 & \text{if } \exists \mathbf{X} \subseteq \mathbf{Att}_i; \exists \mathbf{Y} \subseteq \mathbf{Att}_j; [(\mathbf{X} = \mathbf{Y}) \wedge (\mathbf{X} \rightarrow \mathbf{Att}_i \vee \mathbf{Y} \rightarrow \mathbf{Att}_j)] \\ 0 & \text{otherwise} \end{cases} \tag{5.1}$$

The *joinable* function thus returns 1 only if there is the possibility of lossless join between the answer sets of the input queries. One requirement for this is that at least some subset of the shared attributes be the key of one the sets. This is checked through the closures, $\mathcal{F}^+$, of the queries, or alternatively using the join inference methods previously explained in Sec. 4.2. Of course, the relationship being considered is not precisely a key-foreign key relationship. Such would, as well as a functional dependency, require that each value in one relation find its counterpart in another (i.e., an inclusion dependency). However, due to the fact that the constituent query materializations originate from independent sources, enforcing or requiring inclusion would not be realistic.

Therefore in this simple case, a goal goes on as long as the incoming queries are joinable to the existing ones in the goal. These queries will subsequently be termed *subgoals.* Ultimately, all subgoals are conjoined to assemble the goals.

5.3.1 Composite Subgoals

Composite subgoals are subgoals that are obtained by assembling more than one query in a way other than conjunction. Notice that, due to the commutativity of the conjunction (join) operation, designating a number of queries as constituting a composite subgoal is is no different than designating each one as a subgoal if the composite subgoal is itself created by a conjunction. Hence, I define a composite subgoal as one that is created by the disjunction (or union) of its constituent queries.

In constructing a composite subgoal, a user posing a sequence of queries to several sources is possibly collecting information from each of them in order to create a more comprehensive view of the information. An example would be to send a query selecting for the Italian restaurants in Washington D.C. to two or more different dining registries. The answers obtained are more likely to be unified rather than intersected since the aim here is to garner as many answers as possible.

Another case which requires aggregation rather than cross-referencing is when the complete answer cannot be obtained in a single query due to particular limitations of the data source. The most common reason for this are the *binding limitations* imposed by data sources. In the simplest case, consider an online telephone directory for some large organization. Generally, the service will not have the option of listing all the telephones at once. More likely, the directory will require the user to constrain (bind) at least one field, such as the last name, department &c. A user wanting the whole list will then be forced to ask for one department at a time, or worse a last name at a time. Monitoring such a series of queries, it would be wrong to label them as a conjunction when clearly the intention is to combine the answers into a large list.

The previous example is a special case where the unification of information is associated with a recursive plan. One has to assume that the user knew or obtained the list of departments (which is a table itself) from somewhere, then iteratively expanded each item with the associated telephone listings. Deeper recursive plans may be employed by a user in order to obtain the information needed. Consider a bibliography database that limits binding by requiring that some information about a paper (e.g., title) be given. The only way a user can get all or as many as possible papers is to start with one paper he knows and to recursively expand the citations until no new papers can be found.

Recursive plans may be used in case of poorly structured data services as well. Consider an airline reservation system which only lists direct flights given a source or destination airport. A user trying to get from Washington to Havana will not be able to find a direct flight. Instead, the user will ask for flights out of Washington and will ask follow-up queries asking for possible destinations from those places, until he can recursively find a path to Havana with as few stopovers as possible (e.g., Washington-Toronto-Havana). Again, the aim here is not to join the answers to these queries so they need to be treated differently.

Each example illustrates a different form of recursion requiring a different retrieval strategy. In the case of the bibliography example, a depth first traversal is the best option. In the case of the airline example, breadth first or iterative deepening is needed as depth first recursion will probably be unnecessarily expensive. A third form of traversal is the case where the user recurses using some form of heuristic. Assume the user wants to reach a certain person in a social network service. Similar to the bibliography example, the only way she can accomplish this is to find a string of friends that can connect them. If the user knows that the person lives in a certain city, she may direct the recursion at each level to expand only people living in that city, knowing that this will result in finding a path more rapidly.

Ultimately, there is one common property of recursive query plans that allows one to identify them as such. Each query in such a sequence will bind one of its attributes to a value obtained from a previous sequence. One can formalize this using the intrinsic features of queries previously defined. The feature indicating that two queries Q_i and Q_j may belong to a recursive sequence is given as:

$$recursive(i,j) = \begin{cases} 1 & \text{if } (i<j) \wedge \exists x \in (\mathbf{Att}_i \cap \mathbf{Att}_j); \exists y \in \pi_x(\mathbf{R}_i)\,[(x=y) \in \mathbf{C}_j] \\ 0 & \text{otherwise} \end{cases} \tag{5.2}$$

In creating composite subgoals, a requirement is that the queries that are part of the subgoal have the same "footprint". In other words, it is not possible to unify two answer sets unless they have the same arity and indeed not meaningful unless they share the same attributes. Therefore, a measure is needed to indicate the overlap in attributes between two queries. This can be simply stated as:

$$overlap(i,j) = \frac{\mathbf{Att}_i \cap \mathbf{Att}_j}{\mathbf{Att}_i \cup \mathbf{Att}_j} \tag{5.3}$$

The binary feature associated with overlap is the complete overlap:

$$completeoverlap(i,j) = \begin{cases} 1 & \text{if } overlap(i,j) = 1 \\ 0 & \text{otherwise} \end{cases} \tag{5.4}$$

An analogous and useful feature would compare the overlap in the set of constraints between two queries:

$$constraintoverlap(i,j) = \frac{\mathbf{C}_i \cap \mathbf{C}_j}{\mathbf{C}_i \cup \mathbf{C}_j} \tag{5.5}$$

Thus far, the features I have defined have been completely about the queries themselves. When working in a virtual database environment the provenance of the queries is also very useful. In other words, the source to which the query has been sent and whence the answer originated might further help in defining the role of the query. Particularly if two or more subsequent queries are sent to the same source, there is a higher likelihood that the user is attempting a unification or recursion. In terms of query optimization it is more efficient to ask everything in one query per source. However, if for some reason (e.g., binding limitations or less expressive query mechanism at that particular source) the user is not able to accomplish their subgoal in one query, they will be required to collect the data in pieces. For example, a complete overlap between two queries sent to the same source is a good indicator that the user is combining subsets (e.g., phone numbers in different departments) of the same information. I therefore introduce the following feature:

$$samesource(i,j) = \begin{cases} 1 & \text{if } D_i = D_j \\ 0 & \text{otherwise} \end{cases} \tag{5.6}$$

5.4 Application of Conditional Random Fields

Apart from the complexity of assigning relevance to goals there is some difficulty associated with building the goals in the first place. Several aspects of the sequence will give clues as to the generation of goals:

Do the goals arrive one at a time? The question here is whether the user is done with one task before she starts another one. If this is not true, parts of different goals may arrive interleaved with each other. Without interleaving, the problem of goal identification is simply a problem of boundary detection, namely finding the first query of each goal. Otherwise, goals have to be extracted explicitly.

Do goals evolve coherently? This is related to whether the queries within a goal arrive in the correct order. For the goal to be coherent, each arriving query has to be relevant to at least one other query that is already part of the goal. If the queries arrive out of order, in the worst case, it is possible that there will not exist a coherent goal until the last query of the goal arrives. As an example consider the queries $\mathbf{Att}_1 = \{A, B, C\}$, $\mathbf{Att}_2 = \{K, L, M\}$, $\mathbf{Att}_3 = \{X, Y, Z\}$ and $\mathbf{Att}_4 = \{A, K, X\}$. A goal consisting of these queries will evolve coherently only if Q_4 arrives first. Whether or not this is assumed to be true will determine the solution method.

Are queries used by multiple goals? This factor determines the number of possible goals that can be generated given a sequence of length N. If one assumes that each query can be part of multiple goals, the set possible goals in the query sequence $\mathbf{Q}$ will be the power set of $\mathbf{Q}$, having $2^N - 1$ goals, not counting the empty set. If it is assumed that each query in $\mathbf{Q}$ belongs to just one goal, the set of possible goals becomes the partition of $\mathbf{Q}$. The number of parts in $\mathbf{Q}$ can be at most N, i.e., single-query goals.

Given the above considerations, there is a range to the complexity of the problem depending on which assumptions are adopted. In the simplest case, it can be assumed that queries arrive in order, in a non-interleaved fashion and that each query is part of a single goal only. In this case, the goal identification starts combining arriving queries into a goal. The first query to violate the coherence of the goal will be deemed the beginning of the next goal and the previous goal will be closed.

In the most complicated case, it is accepted that queries can arrive out of order, interleaved with queries of other goals and certain queries might be used by multiple goals. In this case the system will have to keep track of multiple goals at once, evolving some or all of them as new queries are received.

Even if one has a totally objective way of evaluating a given set of queries as to whether they contain an interesting goal, there are still 'technical' difficulties. Given a sequence of queries the set denoting a goal is not necessarily an uninterrupted sub-sequence. Two or more goals may be interleaved with each other or with noise. Some queries in the sequence may be replacements or refinements of older queries. Furthermore, it is possible that the queries needed to build a goal do not arrive in the order they are used. This defeats machine learning methods (e.g., [93]) which are sensitive to the sequence in which queries arrive. Another question is whether the user being monitored is using the same query for multiple goals or one. This distinction affects the number of goals to be evaluated greatly[1].

As the these problems are being discussed, a probabilistic and thus noise-tolerant method of handling the problem has been introduced. The method is distinct in that it can accept realtime streams, constantly evolving a working scenario as new queries arrive. Scenarios that have stopped evolving are dropped for fresh ones as new evidence arrive. Furthermore, the method proposed here leverages the evidence available exclusively in a virtual database environment, namely the source information, to guide its decisions. Such a method will require semantic information about the application domain either in a latent fashion (i.e., machine learning) or explicitly using the constraints of the schema along with a logical inference engine. Such an approach has met with some success in the area of semantic query caching [37] and optimization [22, 58]. The same principle can apply to the present problem as well.

For the purposes of this dissertation, a middle ground between these two extremes is selected. It will be assumed that each query will be part of one goal only. Interleaving will be neglected. The in-order arrival of queries is not a requirement to the extent that the sequence can be kept in memory. As will be seen, the method works in an active window on the sequence starting from the last query going back as far as the window size. A query arriving prematurely can still be added into its goal as long as it is still within this window.

I start by defining a finite state machine which will take one query at a time and transition from one state to another. The state to which the FSM transitions after the arrival of a new query will become the label of that query. The outline of the state machine is given in Fig. 5.1

[1]If the component queries are assumed to be re-used, the number of possible goals increases exponentially, as opposed to a linear growth otherwise.

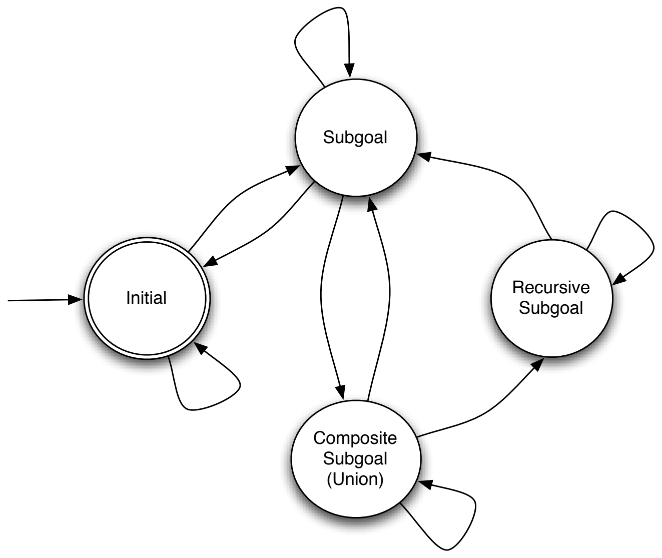


Figure 5.1: **Labeling Finite State Machine**
Only relatively likely transitions are shown.

Converting this finite state machine to a probabilistic one, one arrives at a conditional random field. The probability distribution is thus calculated as given in Eq. 2.1. The sequence $\mathbf{x}$ is the query sequence $\mathbf{Q}$ (i.e., the observations) and sequence $\mathbf{y}$ is the sequence of states, $\mathbf{L}$, the FSM traverses (i.e., the labels). Each transition of the FSM to the *Initial* state denotes the beginning of a new goal. For each arrival of a new query (observation) the CRF is decoded in order to find the label sequence $\mathbf{L}$ that results in the highest conditional probability $P(\mathbf{L}|\mathbf{Q})$. Hence, each time a new query arrives, the conditional probability is recalculated and may therefore change the labels on earlier queries as well.

Figure 5.2 shows an example sequence. Also provided in the figure is the operations performed on the queries by the user. This information is naturally not available to the party monitoring the queries but serve to illustrate the example. Table 5.2 illustrates the labeling for the incoming queries incrementally as each one arrives. The first two queries are joinable, therefore they are labeled with as the initial query and a conjunctive subgoal. The third query initially has no relationship to the preceeding two. It is therefore labelled as the beginning of a new goal. The next few are found to be queries most likely to be unioned with the third query and are thus labelled. The sixth query is again found to be

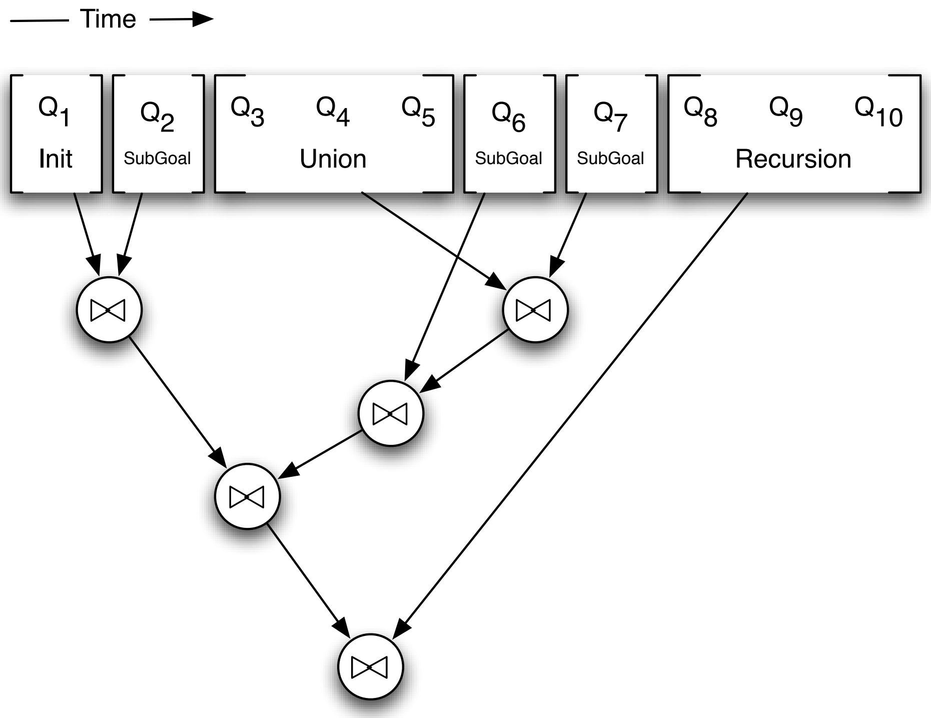


Figure 5.2: **Consolidation of Example Sequence**

unrelated to the existing goals and labelled as the beginning of a new goal. Query 7 happens to be the "keystone" query in that it has join paths to both query 6 and the previous goals. Therefore when query 7 arrives, the labels on the previous queries are changed, coalescing the three goals into one. This is possible since with each new query the labels of all the previous queries are reexamined and a new Viterbi path[2] is calculated for the probabilistic FSM. Subsequently, query 8 arrives and is first deemed to be a single subgoal. However, as more evidence arrives, i.e., queries 9 and 10, it now becomes more probable that the three constitute a recursive subgoal and thus the labels are rearranged.

5.4.1 Features of the Conditional Random Field

In order to complete the definition of the CRF that represents this probabilistic FSM, the set of features $F_k(\mathbf{L}, \mathbf{Q})$ have to be defined. There are two forms of features, namely *state*

[2]A sequence of transitions with highest probability

Table 5.2: **Evolution of Example Sequence**

Time	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8	Q_9	Q_{10}
1	**I**									
2	I	**S**								
3	I	S	**I**							
4	I	S	I	**U**						
5	I	S	I	U	**U**					
6	I	S	I	U	U	**I**				
7	I	S	**U**	U	U	**S**	**S**			
8	I	S	U	U	U	S	S	**S**		
9	I	S	U	U	U	S	S	**R**	**R**	
10	I	S	U	U	U	S	S	R	R	**R**

I: Init, S: Subgoal, U: Union, R: Recursion
New or updated labels are shown in boldface.

and *transition* features. The state features are defined in terms of identity functions. There is an identity function for each label type and each position in the sequence. For example the identity function defining whether the label L_i is the initial state would be:

$$initial(i) = \begin{cases} 1 & \text{if } L_i = INITIAL \\ 0 & \text{otherwise} \end{cases} \tag{5.7}$$

There will be one such identity function for each label per state. For example, given a sequence of 10 queries and the 4 labels defined in the FSM (i.e., Initial, Subgoal, Union, and Recursion), there will be 40 identity functions. Rather, there are 4 state features per clique in the CRF.

The basic building blocks for the transition features have already been given in equations 5.1 through 5.6. I now add two more features that encapsulate the real-time arrival of the queries. The fact that a query arrives after a long delay following the previous query is a valuable observation, particularly in identifying goal boundaries. The first feature is a straightforward timeout feature:

$$timeout(i) = \begin{cases} 1 & \text{if } t_i - t_{i-1} \leq \tau \\ 0 & \text{otherwise} \end{cases} \tag{5.8}$$

where τ is a predetermined timeout, dependent on the application. This is the maximum time alloted to any given session. This is a fail-safe to sever a goal and should be chosen to be a duration that leaves no doubt that the user's interest has expired. For example, for web sessions this is generally set as 2 or 3 days. To achieve a more fine grained use of the time dimension one can also introduce an adaptive timeout:

$$adaptivetimeout(i) = \begin{cases} 1 & \text{if } t_i - t_{i-1} \leq \tau \frac{\sum_{\forall j<i}(t_j - t_{j-1})}{i} \\ 0 & \text{otherwise} \end{cases} \tag{5.9}$$

the adaptive timeout feature is thus true if the delay is larger than a multiple, τ, of the average inter-query delay so far.

Since conditional random fields are log-linear models, in order to capture the query model correctly there is a need to prepare and add conjunctions of the basic features (eqns. 5.1-5.9) into the feature set. For example, a compound feature particularly useful in determining recursion uses the conjunction ($completeoverlap \wedge recursive \wedge samesource$). Up to this point, for each clique in the CRF, there are 9 basic features and 4 state identity features. Along with the possible conjunctions (i.e., the powerset of the 13 features) one may consider up to $2^{13} - 1 = 4095$ features in evaluating the CRF.

Feature Scope

There is also a need to define the scope of each of the features. Recall that a feature function is $f_k(L_i, L_{i-1}, \mathbf{Q}, i)$. However, the functions defined so far take two arguments which are in turn two queries to be compared according to some criterion (e.g., joinability, recursive nature &c.). For each clique, one of these queries is the current one (i.e., Q_i). What this query is being compared to in the sequence is dependent on the scope. One can hence use two different scopes, namely *local* and *global*. The global scope effectively compares all the queries in the range to Q_i and succeeds if there is at least one successful comparison. For example the feature $f^{global}_{samesource}(L_i, L_{i-1}, \mathbf{Q}, i)$ would succeed if there is at least one other query in the sequence that has the same source as Q_i. Therefore, given a basic feature $g_k(Q_i, Q_j)$, a global feature is then:

$$f_k^{global}(L_i, L_{i-1}, \mathbf{Q}, i) = \max_{\forall j \neq i}(g_k(Q_i, Q_j))$$

In contrast, a local feature only checks the queries proximal in the sequence to Q_i. Hence, for example for a window of 5, the local feature would be:

$$f_k^{local}(L_i, L_{i-1}, \mathbf{Q}, i) = \max(\max_{i-5<j<i}(g_k(i,j)), \max_{i<j<i+5}(g_k(i,j)))$$

Note, however, that the state features previously mentioned and any conjunctions containing them cannot be global and can only be local within a window of 2. Otherwise, the probability of a given label would not be independent of previous states except the neighboring one.

Time-shifted Features

The final enrichment to the feature space involves the *time-shifting* of features, such that clique potentials can be affected by previous and future observations. A time shifted feature $f_i^{<j>}$ would be defined as:

$$f_i^{<j>} = f(Q_{i+j}) \tag{5.10}$$

So, a feature $f_i^{<-2>}$ would be the said feature for the observation two queries previous to Q_i used to evaluate the potential for the i-th clique in the CRF. Again, this is limited to the transition features since the potential of a given clique cannot be a function of the label of state not neighboring it. Therefore, features containing any of the state identity functions can only be shifted by -1. According to these restrictions and assuming time shifts of -2, -1, 0, 1, 2, global and local forms for all the possible features, one may have up to 12278 features per clique.

5.4.2 Gain-based Feature Selection

Note that the number of features is the combinatorial maximum of features obtained in the model thus far. All of these features are not expected to be predictive to the same degree. In fact, depending on the case, some might be completely useless. Hence the need to prune those that have little utility in order to keep the model manageable and more importantly to avoid over-fitting the training data.

In order to do this I use gain based feature selection. Recall that the training of a CRF involves optimizing the weight parameter vector λ in the log-likelihood function (Eqn. 2.2) in order to obtain the maximum likelihood. This is done in a supervised manner by iteratively maximizing for a set of training data. Each training sequence of queries has an associated sequence of correct labels which are used to optimize the parameters, λ, to obtain the weights that will have the least training error. Since some of the features are expected to be less discerning, the changes in the associated weights will have less impact on the training error. The idea is termed *Gain-based Feature Selection* the method for CRFs is described detail in [67]. The gain of a feature f is defined as:

$$G_\Lambda(f) = \max_\mu(\mathcal{L}_{\Lambda+f\mu} - \mathcal{L}_\Lambda)$$

where $\mathcal{L}_{\Lambda+f\mu}$ and $\mathcal{L}_\Lambda$ are the log-likelihoods of the CRF with or without the feature f, given the training data. If done naïvely, this gain calculation will be very time consuming, given the large number of features. Fortunately, there are several optimizations including mean field approximation and limiting the gain calculation only to mislabeled outputs.

These are quite outside the scope of this work and I refer the reader to [67] for an extensive treatment of the subject. Using this gain based pruning I show that one is able to use a tenth of the features of the original model while gaining better accuracy and better generalization (Cf. Sec 8.1).

5.5 Summary

Up to this chapter, the discussion has assumed that there is a set of selected queries that constitute the components of a consolidation. In actuality, queries arrive and are monitored as sequences rather than discrete sets. There is no clear delimitation of the point at which one task ends and a new one begins.

This chapter explores this issue and introduces a method to segment ongoing sequences into clearly delimited 'tasks' using the concepts from previous chapters along with the application of conditional random fields to probabilistically labeling incoming queries. Conditional random field models are supervised learners which are trained with pre-labeled data. The model resulting from the training is used to label future sequences. The labels serve as delimiters for tasks and also classify component queries within a single task as to their function and whether they should be conjunctively or disjunctively assembled to form a consolidation.

The chapter starts by defining the intrinsic features of each query in a sequence and goes on to derive the features relevant to sequence analysis, scoping and time-shifting of these features and discusses the use features dependent on the content of the queries. Finally the training of the CRF model is described with attention to regularization of the objective function and pruning of features.

Chapter 6: Enhancement of Candidate Consolidations

In Chapter 4, I described and discussed the generation of minimal consolidations as a limited set of solutions which would form a basis of joinability that was later used in Chapter 5 as one of the features in isolating coherent goals from sequences of queries.

Having isolated a goal, created minimal consolidations and ranked them, one can further enhance these solutions with information from the schema and from earlier patterns of user activity. I approach the enhancement of minimal consolidations in two dimensions:

- **Horizontally:** Given the projection of a minimal consolidation on the global schema, one could leverage the functional dependency information available as part of the global schema to add more attributes to the consolidation. This might increase the amount of information available to the monitor, thereby capturing additional details that may have not been requested explicitly by the querying agent.

- **Vertically:** The tuples in the extension of a minimal consolidation are the result of equi-joins and not necessarily as precise as the consolidation the querying agent had in mind. As discussed previously, the querying agent may indeed be applying additional selection criteria on to the results at the integration site. These selections would not be visible to the monitor (see Fig. 3.1) due to their global nature. However, they can be estimated by analyzing trends in earlier query activity.

6.1 Expanding the Projections

The minimal consolidation of a set of queries $Q_1, \ldots, Q_n$ is a join plan that assumes that the queries themselves are sufficient for the goal of the querying agent. In reality, this may not be the case. The querying strategy of the agent may include outside information that is used to come up with the query strategy.

Example 6.1. Assume the most likely consolidation of a set of queries is the following:

Make	**Model**	**Year**
IBM	RS/6000	1990
Apple	Xserve	2002
IBM	AS/400	1988

This particular extension may not be very informative by itself. Assume the global schema has the following functional dependency:

$$\{Make, Model, Year\} \longrightarrow \{Processor\}$$

The initial extension can be then expanded with the appropriate join to arrive at:

Make	**Model**	**Year**	**Processor**
IBM	RS/6000	1990	PowerPC
Apple	Xserve	2002	PowerPC
IBM	AS/400	1988	PowerPC

Clearly, this expanded version of the consolidation offers more clues as to the aim of the querying agent (i.e., searching for computers with a PowerPC processor). The reason the processor attribute was not used might be because the local source did not have that attribute, or that it could not be used as selection criterion, or that the user did not know that the same information could be gathered using the processor field. □

This expansion of projections can be done using the functional dependency closure Σ^+ assumed to be present along with the global schema $\mathcal{G}$. Figure 6.1 gives a reference

Input: Query Q, set of attributes $\mathbf{A}$
Output: Expanded query Q
Environment: Global relation $\mathbf{G}$, FD closure Σ^+

```
1:  $\mathbf{A}^+ \leftarrow \mathbf{A}^2 - \{\emptyset\}$
2:  for all $\mathbf{X} \in \mathbf{A}^+$ do
3:    for all $F \in \Sigma^+ \mid F : \mathbf{X} \longrightarrow \mathbf{Y}$ do
4:      if $\exists y \in \mathbf{Y} : y \notin R$ then
5:        $R \leftarrow (\Pi_{X \cup Y}(\mathbf{G}))$
6:        if R is materializable then
7:          $Q \leftarrow (Q \bowtie_{\mathbf{Y}} R)$
8:          $Q \leftarrow Expand(Q, \mathbf{Y})$
9:        end if
10:     end if
11:   end for
12: end for
13: Return $Q$
```

Figure 6.1: **The Expand Algorithm**

algorithm that will recursively expand an extension given a set of attributes to expand on. Therefore, any consolidation Q may be expanded by executing $Expand(Q, \mathbf{Att}_Q)$. The expansion algorithm assumes that the global schema $\mathcal{G}$ is acyclic and can thus be reduced to a universal relation G and that any subschema (denoted R in line 5 of the algorithm) of the universal relation may be materialized by sending the decomposing to relevant local sources (as performed in line 6).

Clearly, this may not be the case in some situations. There may be cycles in the schema or some parts of the schema may not be materializable. In the case where the materialization of an attribute fails, that branch of the recursion, and any transitive expansions resulting from thereon, may fail.

In the case of a cyclic schema the algorithm is still guaranteed to halt since ultimately either the dependency closure will be exhausted or there are no more attributes that can be added to Q (check in line 4). However, the expansion based on a cyclic schema will not necessarily be correct.

6.2 Adding Global Selection Constraints

Once minimal consolidations are generated, there is at least one relation representing the way queries sent to different sources may have been joined. This relation contains all the information collected by the querying agent and thus, in a way, is an acceptable result. At least, in terms of information disclosure, this relation, provided that the join scenario is correct, is complete. However, in terms of the precision of the answer, more processing might be needed.

Example 6.2. Consider a user asking for the following views from two different sources:

- $Q_1 = \{\text{SSN, Year, TaxOwed}\}$
- $Q_2 = \{\text{SSN, Year, TaxWithheld}\}$

The top ranked join will probably be $Q = Q_1 \bowtie_{SSN} Q_2$. The relation **Q**, while holding all the information retrieved by the user, is not really very informative at first glance. Knowing something about the domain of the information one might easily speculate that the user would like to further refine this data offline by comparing the amounts of tax owed and withheld by each taxpayer in the table by year. The result would become:

$$Q' = \sigma_{Year_1=Year_2}(Q_1 \bowtie_{SSN} Q_2) \tag{6.1}$$

□

Notice that, due to the distribution of the information among the two sources, the querying agent could not have done the comparison locally, and is forced to do it at the integration site. Therefore, as an observer only able to monitor retrievals, one cannot say for certain that it is performing such a refinement.

Example 6.3. The previous example can be further extended by speculating that querying agent wants to see only the taxpayers who got a refund. So the global question now becomes:

$$Q'' = \sigma_{Year_1=Year_2, TaxOwed<TaxWithheld}(Q_1 \bowtie_{SSN} Q_2) \tag{6.2}$$

Again, this comparison would have to be done at the integration site and thus would be invisible to the monitor. □

Obviously, while contained in the original join Q, the final question Q'' is considerably different. One would like to be able to infer such constraints added at the integration site. It should be noted that since the monitor has no information as to the operations performed offline, there is no certain way of telling precisely what the user wanted to obtain.

The information one does have, namely the projections done, however, allow educated guesses as to the use the information will be put to, once combined. While join inference can be done in a domain-agnostic manner, the same is unfortunately not possible for this problem. Consider the previous examples; there is no way to speculate that a comparison on the year field or a comparison of the tax owed to the tax withheld is probable without having some domain knowledge on the subject of taxes.

Since encoding such domain knowledge manually is prohibitively tedious and expensive, one would like to distill near equivalent knowledge by mining existing query logs. Assume there is a large number of global queries available from previous usage of the information sources. Also assume that these global queries also carry information on the operations performed offline. Such a repository of existing queries would contain the knowledge required, albeit in a latent fashion. From now on, I shall call this body of existing global queries the *training set.*

Referring back to the tax example, if there are a significant number of examples in the training set which project the set of attributes { TaxOwed, TaxWithheld } and a significant portion of these also contain the selection predicate $TaxWithheld > TaxOwed$ one can infer the rule:

$$\Pi(TaxWithheld) \land \Pi(TaxOwed) \rightarrow \sigma(TaxWithheld > TaxOwed) \tag{6.3}$$

The problem of inferring a set of rules such as the one above from a set of transactions is an active field of data mining called *association analysis* where the transactions are analyzed in order to discover frequent item sets [8, 9]. The classical example is market basket analysis where purchase records of retailers are mined in order to find out which products are purchased together (e.g., beer and peanuts).

In analogy, in this case, I mine the training set of queries to find out which attributes are frequently projected together. Furthermore, I would like to see whether the fact a set of attributes is projected also implies the existence of a comparison.

The frequency of a set of items in the training set is the *support* of the item set. For example, if the attributes TaxWithheld and TaxOwed are projected together in 30% of the queries in the training set, the attribute set {TaxWithheld, TaxOwed} is said to have a support of 0.30. Note that the support of an attribute set is *anti-monotone*, so that it cannot have a higher support than any of its subsets. In other words, if TaxWithheld is projected in 40% of the queries in the training set and TaxOwed is projected in 30%, the set {TaxWithheld, TaxOwed} cannot have a support greater than 0.30.

Algorithms that mine for frequent item sets generally depend on this property to prune the superset of possible attributes so that most subsets of attributes do need to be considered.

One can therefore mine the training set of queries for sets of projected attributes that have at least a certain degree of support. The selection of a threshold for support depends on many factors including the total number of attributes the in the domain, the size of the training set and the extent of generality to which one would like to capture the knowledge [88]. In this case, the support level is selected such that a set of projected attributes is above a absolute number of occurrences. Therefore, the support level is set depending primarily on the size of the training set. As an example, if there is a training set of 500 queries, a minimum support of 0.1 would mean that the set of attributes occurs at least 50 times. This entails a statistically significant amount of interest in those attributes. Therefore, as the training set gets larger, one may theoretically decrease the minimum support level, such that the absolute number of occurrences is statistically significant. However, in reality, as the support threshold gets smaller, the number of rules generated and the time required to train get larger. Hence, the support threshold to be selected is essentially a compromise between the desire to select the minimum threshold to ensure statistical significance and real world considerations of time complexity and rule-base size.

Once a threshold is set, a standard algorithm such as those mentioned previously may be used to find maximal frequent itemsets. At this point there is a minor difference between the present problem and that of market basket analysis.

Consider a market basket such as:

$$\{Beer, Milk, Diapers\} \tag{6.4}$$

having a support of, say 10%. Standard rule generation partitions this frequent itemset to generate a rule such as:

$$Milk \wedge Diapers \rightarrow Beer \tag{6.5}$$

where the probability of beer existing in a basket is above a certain threshold (i.e., minimum confidence). A query in a training set is analogous to a basket in a database of transactions. However, in the case of queries, there are two distinct subsets associated with each query, namely the set of projected attributes and the set of constraints. Thus the following SQL query for example:

```
SELECT A1, A2
FROM R
WHERE C1 AND C2;
```

would be a record in the training set in an ordered-pair form:

$$\{\{A_1, A_2\}, \{C_1, C_2\}\} \tag{6.6}$$

Ultimately, one requires rules that have a conjunction of elements from the first part of the pair in the antecedent and elements of the second part in the consequent, e.g.,

$$A_1 \wedge A_2 \rightarrow C_1 \tag{6.7}$$

Therefore, any itemsets that contain only comparisons (i.e., C_i) are useless. This allows the reduction of the search space for frequent itemsets even further, in additon to standard pruning techniques of standard algorithms mentioned in the background chapter. Consider the search tree given in Figure 6.2. This tree maps out how the search for frequent patterns progresses.

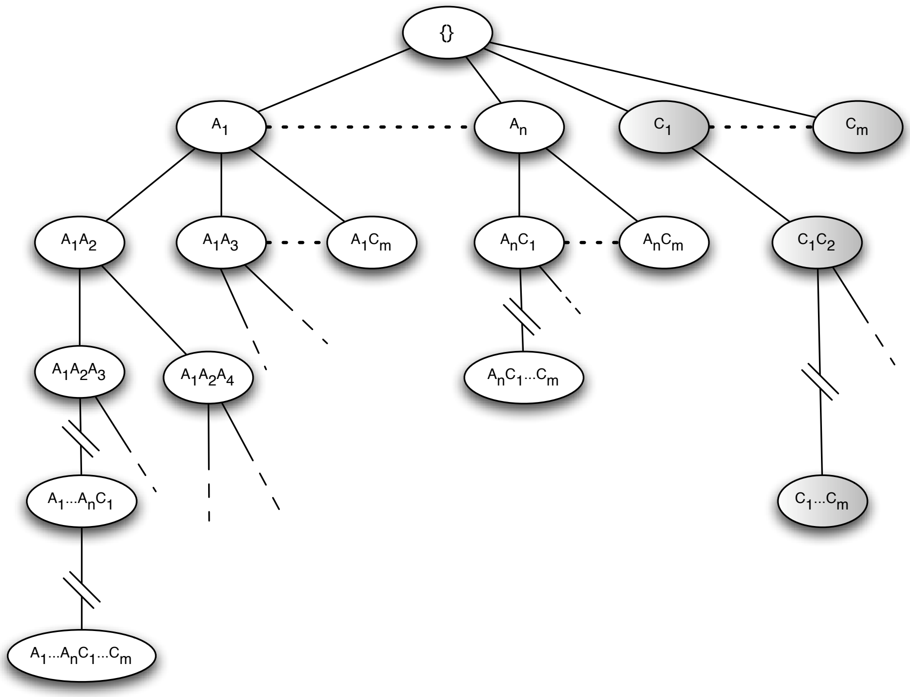


Figure 6.2: **Partial Evaluation of Lattice**
Prefix tree shows the search space of a depth-first search. A_i are projected attributes, C_i are comparison predicates. The grey shaded nodes are pruned away before searching.

I adopt an ordering function such that all projected attributes and comparisons are ordered lexicographically within themselves and all projected attributes are smaller than any comparison predicate. The ordering is thus:

$$\begin{aligned} A_i &< A_{i+1} : (1 \leq i \leq n) \\ C_i &< C_{i+1} : (1 \leq i \leq m) \\ A_n &< C_1 \end{aligned}$$

Under these circumstances one can prune a good portion of a prefix-ordered search tree. Namely, none of the subtrees that are rooted at a C_i node need be visited, since none of these trees will have any nodes which contain any projected attributes, A_i. Such nodes will only lead to rules without antecedents and are therefore not needed. In Fig. 6.2, these nodes are shaded gray.

Note that such pruning is only really viable in a depth-first search. A breadth-first searching algorithm generates all 1-itemsets before moving on, which is precisely what is to be avoided. Furthermore, k-1 itemsets are used when evaluating k-itemset candidates, hence one cannot get away with pruning half the 1-itemsets, since all of them are required to build 2-itemsets which are in turn required to build 3-itemsets and so forth.

Once a list of maximal frequent itemsets are generated, these lists are filtered such that only those that include at least one comparison item are left. Just as there is no point in having rules with no antecedent (hence the pruning mentioned above), there is no point in having rules without consequents. Therefore, itemsets with only projected attributes in them are useless.

With the remaining itemsets, the rules are generated much like standard rule generation in association rule mining. The itemset is partitioned into a rule such that given the set of items in the antecedent (i.e., a set of projected attributes), the queries in the training set have at least a miniumum probability of having a given consequent (i.e., a comparison or conjunction of comparisons). Given an itemset such as $\{A_1, A_2, A_3, C_1, C_2\}$ the rules to consider would be:

Rule 1 $A_1 \wedge A_2 \wedge A_3 \rightarrow C_1$

Rule 2 $A_1 \wedge A_2 \wedge A_3 \rightarrow C_2$

Rule 3 $A_1 \wedge A_2 \wedge A_3 \rightarrow C_1 \wedge C_2$

Notice that, in analogy to support, confidence has an anti-monotone property as well under these conditions. In the example, it is clear that rule 3 cannot have a confidence higher than rule 1 or rule 2. Therefore, one may generate rules in a bottom-up manner by starting with single comparisons and later combine those with confidence above the minimum into larger rules involving conjunctions on the consequent side. The same kind of pruning as done in Apriori and other breadth-first methods are applicable due to this anti-monotone property. Since one would like to keep the rule base small, I only use maximal rules. In other words if rule 3 in the above example meets the minimum confidence, rules 1 and 2 need not go into the rule base as they are subsumed by rule 3.

Once the rule base is constructed, it can be used for query enrichment. Whenever a set of query fragments are joined, the projection set of that join can be compared against the rule base and additional comparisons may be suggested to the user. The consequent of each rule that is satisfied by the projection of a join is suggested as a possible enhancement, ranked by the confidence of the rule.

6.3 Summary

In this chapter I have suggested two methods by which likely minimal consolidations may be enriched to approximate the consolidation done by the querying agent at its integration site.

The first enhancement involves traversing the functional dependency closure in order to expand the projection of the minimal consolidation. This expansion is aimed to estimate the ‘offline’ information the querying agent may have and will in turn aid in finding a more concise intensional expression to encapsulate the consolidation, as will be discussed in the next chapter.

The second enhancement is the addition of global selection constraints to the consolidation based on the estimation of which global selections are likely. This estimation is done by using a set of rules linking the existence of certain attributes in the projection to the likelihood of the constraint to be introduced. These rules are generated by performing association analysis on a training set of global queries in the domain where both the projections and the global selections are visible.

Chapter 7: Re-Encapsulation of Candidates

The problem of intensional answering has long been an open research area [71]. These systems seek to generate a concise expression that defines the information in a database or a subset thereof. The aim is to derive the queries or schemata that completely and precisely define a database relation or a view of the database by processing only the extensional representation (i.e., the collection of tuples that comprise the view or relation). The problem is closely related to data mining in that it seeks to derive the underlying explanation from a collection of data. However, the real use of the system is in information integration. The ability to generate intensional and machine parseable expressions from external views allows systems to automatically integrate new information sources into a virtual database [16]. An additional application also arises in database security, where one wants to analyze the intensional representation of the records received in order to determine if an attacker is trying to infer restricted information [2].

This chapter describes the method [3] developed for generating such intensional expressions for the consolidations generated using the approach detailed thus far in the dissertation. In conjunction with projection expansion explained in the previous chapter, re-encapsulation of the extensions of consolidations with new intensional expressions may yield simpler explanations to the consolidated query.

Example 7.1. Let there be an consolidation with the original intensional expression:

$$\begin{aligned} Q &= (\sigma_{(AreaCode=212)}(Phonebook) \cup \sigma_{(AreaCode=917)}(Phonebook)) \\ &\quad \bowtie (\sigma_{(Job=\text{`CEO'})}(Employee) \cup \sigma_{(Job=\text{`VP'})}(Employee)) \end{aligned}$$

or, “The vice presidents and CEO who have area codes of 212 or 917”. Let this query be materialized as:

ID	**AreaCode**	**Phone**	**Name**	**Job**	**Salary**
1	212	555-1234	Alice	CEO	\$750,000
43	917	555-2355	Edward	VP	\$540,000
56	212	555-4322	Jane	VP	\$620,000

Analyzing at the extension, a re-encapsulation might find the following intensional expression:

$$\begin{aligned} Q &= (\sigma_{(AreaCode=212)}(Phonebook) \cup \sigma_{(AreaCode=917)}(Phonebook)) \\ &\quad \bowtie \sigma_{(Salary \geq \$500{,}000)}(Employee) \end{aligned}$$

or, “Employees making more than \$500,000 and having area codes 917 or 212”. Now, consider the same query with its projection expanded as described in Sec. 6.1:

ID	**AreaCode**	**Phone**	**Name**	**Job**	**Salary**	**City**
1	212	555-1234	Alice	CEO	\$750,000	New York
43	917	555-2355	Edward	VP	\$540,000	New York
56	212	555-4322	Jane	VP	\$620,000	New York

A re-encapsulation of this extension might yield:

$$Q = (\sigma_{(City=\text{`New York'})}(Location) \bowtie \sigma_{(Salary \geq \$500{,}000)}(Employee)$$

or, “Employees in New York who make more than \$500,000”. Clearly, the last encapsulation is much more concise and arguably more informative than the original. □

The method I will describe in this chapter is essentially a locally optimizing search which employs genetic programming. Candidate queries which attempt to explain a set of records are generated. These candidates are evolved and recombined with each other until a satisfactory result is obtained. The attributes and attribute values used for these candidate queries are selected using naïve Bayesian inference which incorporates the probability distributions of various values occurring in the database.

7.1 Overview of Method

There are two primary ways one might use the programs resulting from the evolutionary process of genetic programming. In the first case, the point of interest is the performance itself and one is only interested in having a program which performs the required function without necessarily needing to know the inner workings. The second case, the object is the structure of the program itself. One would like to evolve a program that performs a certain function and subsequently would like to see the mechanism by which it performs. The latter case is the approach of this chapter.

A query is, in effect, a computer program that extracts a certain subset of some master relation as a separate class. The point of interest is the mechanism that caused the original query to pick out a given subset. Therefore, the method evolves programs (i.e., queries) that perform the same function as the original query. Once the evolution has succeeded in producing an identical or close enough extension, one can examine the mechanism by which it operates. This mechanism is the intensional re-encapsulation one is looking for.

In order to describe the operation of the system I provide a simple example and will follow through the rest of the chapter with this example. The master extension (hereafter called $\mathbf{R}$) is given in Table 7.1. The set of records retrieved (i.e., the extension of some consolidation for which the system attempts to find the intensional answer) is called the target record set ($\mathbf{T}$ s.t. $\mathbf{T} \subseteq \mathbf{R}$) and is given in Table 7.1.

The main premise of the approach is that it creates a population of candidate queries and evolves the population until one or more of the candidates perform as required. The

Table 7.1: **An Example Master Extension**

Employee	**Job**	**Dept**	**Salary**
Alice	Engineer	Clothing	12430
Bob	Salesperson	Hardware	14500
Carol	Accountant	Food	15340
David	Engineer	Hardware	11230
.	.	.	.
.	.	.	.
.	.	.	.

Table 7.2: **An Example Target**

Employee	**Job**	**Dept**	**Salary**
Carol	Accountant	Food	15340
George	Accountant	Furniture	9540
Mike	Accountant	Hardware	12000

candidates are each queries Q_i. The performance requirement here is that $\mathbf{Q}_i \equiv \mathbf{T}$. Note that this is the ideal requirement and in the real system it could be that an amount of similarity might be deemed sufficient instead of absolute equivalence.

The method starts with the generation of a number of candidate Q_i's. This initial generation is said to be random in that it is a set of arbitrary queries. They are, however, each fully operational, correct queries made out of primitives which have been selected probabilistically to have a higher chance of being fit to the task at hand. The primitives themselves, their selection and combination into queries will be discussed in detail later in the chapter.

Each candidate in this initial population is then evaluated to obtain a fitness value. The fitness in question can be defined as the similarity of $\mathbf{T}$ to the extension $\mathbf{Q}_i$, produced by materializing (i.e., executing) the query Q_i. The similarity measure will be discussed in a later section but generally is a value between zero and one where the former means a totally disjoint $\mathbf{T}$ and $\mathbf{P}_i$ and the latter denotes $\mathbf{P}_i \equiv \mathbf{T}$.

The candidates with the highest fitness are then bred with each other using either direct combination, crossover or mutation to generate a new population of candidates. The

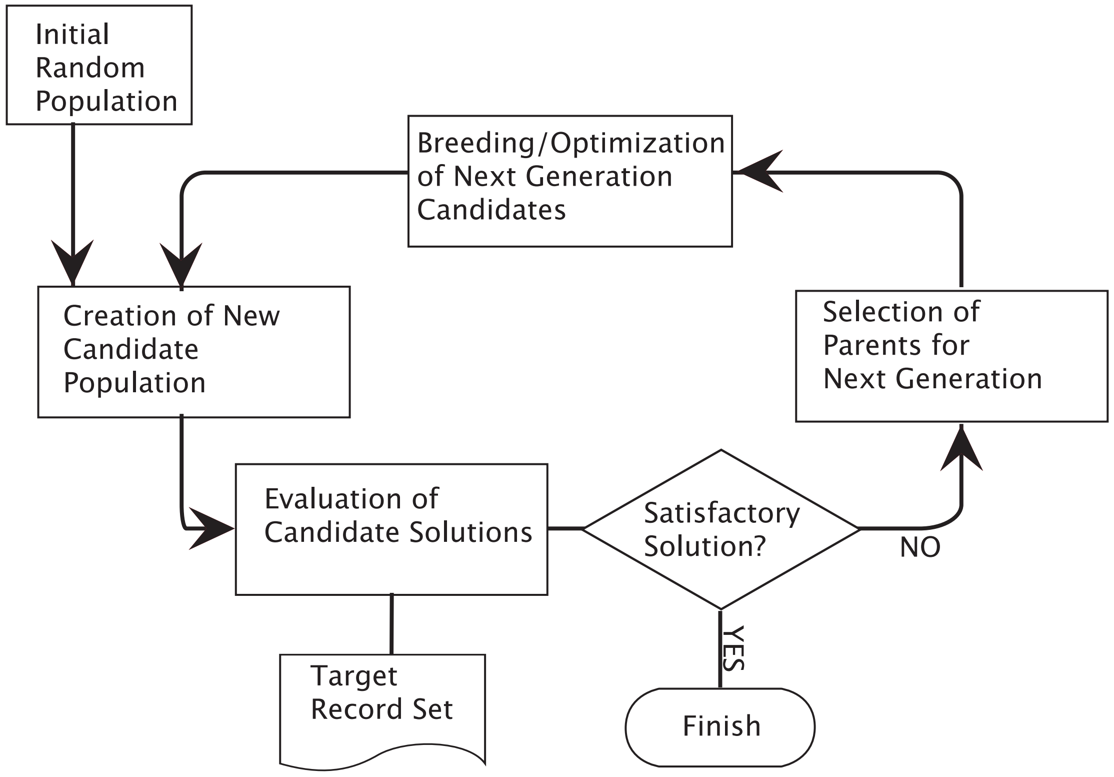

Initial Random Population
Breeding/Optimization of Next Generation Candidates
Creation of New Candidate Population
Selection of Parents for Next Generation
Evaluation of Candidate Solutions
Satisfactory Solution?
NO
YES
Target Record Set
Finish

Figure 7.1: **Overview of the Method**

selection of high fitness candidates and the breeding process that is used to obtain the new generation of candidates is a discussion to follow. This new generation is then re-evaluated in the same manner as its predecessor and the evaluation is used to produce yet a newer generation of candidate population. This evolution gradually increases the mean fitness of the candidates in the population. The cycle is stopped once one candidate in a population achieves a satisfactory fitness. A satisfactory fitness is defined as a similarity to **T** above a certain limit which may be preset as a constant or as a function of the sizes of **R**, **T** or depending on the application. Figure 7.1 gives a schematic representation of the cycle. The cycle is unbound in that it will never stop until the threshold is reached. Therefore, as a practical consideration, generally an exception is added which stops the evolution if there is no improvement over a given number of generations.

7.2 Generating the Initial Population

In keeping with the premise of genetic programming, the queries $\mathbf{Q}_i$ in the population are represented as trees. These trees can be parsed directly to yield relational algebraic expressions. Each tree has selection operators as terminal nodes and set operators as internal nodes.

The system has three fixed set operators for use in queries. These are:

Union $\cup(A, B) = \{\forall x : x \in A \vee x \in B\}$ Arity: 2

Intersection $\bigcap(A, B) = \{\forall x : x \in A \wedge x \in B\}$ Arity: 2

Set Difference $\backslash(A, B) = \{\forall x : x \in A \wedge x \notin B\}$ Arity: 2

Apart from the basic operators the system has automatically defined selection operators which are created with respect to the database and target in question. The following classes of automatically defined operators, presented as relational algebra constructs, are created:

EQSELECT $EqSelect[a, v_i](\mathbf{R}) := \sigma_{(a=v_i)}(\mathbf{R})$

NUMSELECT $NumSelect[a, v_i, v_j](\mathbf{R}) := \sigma_{v_i \leqslant a < v_j}(\mathbf{R})$

LIKESELECT $LikeSelect[a, v_i](\mathbf{R}) := \sigma_{a \supseteq v_i}(\mathbf{R})$

In the preceeding descriptions a denotes an attribute in the master relation and v denotes a distinct value of that attribute. Thus, instances of these operators may be created for each possible combination of values and attributes. Note that these operators are selected with care to be easily represented in SQL. They map directly to the SQL phrases for easy construction of queries:

$EqSelect(a, v_i) \rightarrow$ "WHERE $a = v_1$"

$NumSelect(a, v_i, v_j) \rightarrow$ "a BETWEEN v_1 AND v_2"

$LikeSelect(a, v_i) \rightarrow$ "a LIKE '%v_1%' "

Given the example target in Table 7.1, for instance, an operator such as $EqSelect(Job, `Engineer')$ would make little sense in finding a proper intensional answer for the target since the target is composed solely of accountants. Therefore, while generating individuals one has to assign probabilities to each attribute and value with respect to their relevance to the target extension.

The question then is how to assign probabilities to operators. This is done in different ways for different types of attributes. For the purposes of this dissertation, I define three types which map to the three different selection operators above. These are

Nominal Attributes Attributes made of short strings with enumerable ranges of values or integers with no ordinal relationship. These are exemplified by database fields such as model numbers, names, phone numbers &c.

Numeric Attributes Numbers with possibly infinite range which also have an ordinal relationship such as salary, size, distance, time &c.

Text Attributes Long strings. Examples are comment fields, addresses &c.

The nature of each attribute (or field) in a database relation is assumed to be known at the outset. Given the type of each attribute the system can perform feature selection for the genetic programming process. Whenever a new operator is needed during the evolutionary cycle, either during initial population generation or subsequent mutations, a closure[1] is generated depending on the type of attribute. The closure is the combination of a function and its lexical environment. The function to be used is determined by the attribute such that it is an *EqSelect* if the attribute is nominal, a *NumSelect* if the attribute is numeric and a *LikeSelect* if the attribute is a text attribute. The lexical environment on the other hand is characterized by the values of a, v_1 and v_2, if applicable.

At this point, consider a random variable A which has a range of possible values equal to the set of possible attributes of the master relation $\mathbf{R}$. For reasons of simplicity, let this be a discreet random variable with uniform distribution, such that the value of A will be equal to any of the attributes with equal probability. Therefore, while generating closures one can use the distribution of this variable to determine the value of a while building closures as needed. Once the attribute for the closure is determined, depending on the type of attribute I define the values of v_i in a similar manner.

7.2.1 Nominal Attributes

A good portion of the attributes in a database are probably going to be nominal values. These are defined as values which have no inherent order. Examples are names, boolean values, pointers to other tuples, &c. Given a nominal attribute in $\mathbf{R}$ called a_i, I now define a discreet random variable A_i. The range of this random variable is the set of values associated with the relevant attribute. In the example of Table 7.1, the range of A_{Dept} would be { 'Clothing', 'Food', 'Furniture', 'Hardware' }. To determine the probability of each value I use Bayesian methods. For each value v_j of attribute a_i in $\mathbf{R}$ I define the prior probability as:

[1]The term closure in this context is meant in the sense it is used in functional programming literature.

$$p(v_j) = \frac{|\sigma_{a_i=v_j}(\mathbf{R})|}{|\mathbf{R}|} \tag{7.1}$$

This is essentially the probability of each value occurring in the master relation and can be calculated once and stored as long as the master relation does not change. In a virtual database, the calculation of this prior would in essence mean materializing the whole global schema **G**. Clearly such an operation my be intractable. Thus the prior probabilities can be instead estimated with sampling and stored. Next, the prior probability of a record being part of the target subset is:

$$p(\mathbf{T}) = \frac{|\mathbf{T}|}{|\mathbf{R}|} \tag{7.2}$$

Again the sizes of the relations which constitute the basis for any one consolidation may not be known and may have to be estimates. The conditional probability of the value v_j given the subset **T**, however, can be calculated directly from the extension available:

$$p(v_j|\mathbf{T}) = \frac{|\sigma_{a_i=v_j}(\mathbf{T})|}{|\mathbf{T}|} \tag{7.3}$$

The preceding two calculations are done whenever a new target extension (i.e., a consolidation) is processed and involve computing the results of a single SQL query of the form:

SELECT DISTINCT a_i, COUNT(a_i) FROM **T** GROUP BY a_i;

Once these values have been calculated the probability of a record being part of the target given evidence that its attribute a_i has a particular value v_j may be derived using the Bayes rule:

$$p(\mathbf{T}|v_j) = \frac{p(\mathbf{T})p(v_j|\mathbf{T})}{p(v_j)} \tag{7.4}$$

This probability happens to be a good indicator for predicting the usefulness of a particular value for the purposes of defining the target $\mathbf{T}$ intensionally. Thus I use this conditional probability in assigning the probabilities of the random variable A_i. Note that since A_i is likely to be very sparse[2], the conditional probabilities cannot be used directly. So I define A_i using the normalized values:

$$p(A_i = v_j) = \frac{p(\mathbf{T}|v_j)}{\sum_k p(\mathbf{T}|v_k)} \tag{7.5}$$

Therefore, one can define the selection value of all the selector closures for nominal attributes using the random variables A_i.

7.2.2 Text Attributes

Since text attributes are character strings of arbitrary length, it would not be wise to use the above method for these directly. Defining a conditional probability for all possible values of a long string would be both expensive and also futile in that it would not give the pattern of occurrence. Instead, one can think of strings as combinations of tokens out of a master dictionary (i.e., a bag of words).

One can start by consdering all the values of a given text attribute of $\mathbf{R}$. Each value is tokenized in that it is represented as a set of tokens. A token is defined as a character substring delimited by whitespace or punctuation characters. The tokens can be sampled from the materialization of the virtual database $\mathcal{G}$, or if that is not tractable they can be generated from a wordlist of the language in question. Once the tokens and their frequencies are tabulated one can calculate the prior probabilities of each of the tokens similar to the way it is done in Eqn. 7.1. Again, like nominal values, these prior probabilities have to be calculated only once unless the data sources underlying the virtual database change dramatically.

[2]i.e., most values a_i will have zero probability

When re-encapsulating a new target, one may calculate the conditional probabilities of tokens occurring in the target in a fashion similar to Eqn. 7.3. The prior probability of the target itself is defined as the ratio of the total frequency of tokens occurring in the target to the total frequency of tokens in the master dictionary. Hence, one can generate a random variable T_i which has a range composed of all the tokens. The probability of T_i being equal to a particular token would be equal to the normalized conditional probability of a record being part of $\mathbf{T}$ given that the token occurs in the text attribute t_i of that record. Thus, one can use random values of T_i to define selection values for the LikeSelect closures.

7.2.3 Numerical Attributes

Numerical attributes can also be given intelligent selection values using random variables. Ideally this would be done using a continuous random variable with values that closely mimics the probability density function of the distribution of the numerical attribute in question. This however, is a rather complex approach that requires a good deal of information about the distribution of values within the target extension as well as the source database. Since such information is clearly very tedious or impossible to generate, I opt to reduce numerical attributes into nominal attributes using discretization.

The range of values occurring in a numerical attribute of the source database $\mathbf{R}$ maybe divided into discreet intervals which capture the semantic essence of the data. In a healthy discretization, the semantic difference among values falling into separate intervals is significant but variation within intervals is not. Arguably the best way to define such intervals on a numeric attribute is by using human experts who are well appraised of the meaning of the data.

In the absence of such help, one can still define a good enough method to discretisize the data. The method I use is inspired in most part by [77] and uses a recursive algorithm to split numerical attributes into discreet intervals. The objective of the algorithm is to split the range of an attribute into intervals that have variation below a certain threshold. The algorithm is given in Figure 7.2.3.

```
For a given numeric attribute $n_i$: **procedure** Bisect
**Input:** $V \leftarrow SortAscending(v_i \in n_i)$, Variation Threshold $dV_{max}$, p, q
**Output:** Set P of cut points
 1:
 2: $P \leftarrow \emptyset$
 3: **if** $V[q] - V[p] \leqslant dV_{max}$ **then**
 4:     return P
 5: **else**
 6:     $P \leftarrow V[(q-p)/2]$
 7:     $P \leftarrow P \bigcup Bisect(V, dV_{max}, p, (q-p)/2)$
 8:     $P \leftarrow P \bigcup Bisect(V, dV_{max}, (q-p)/2+1, q)$
 9:     return P
10: **end if**
```

Figure 7.2: **The Bisect Algorithm**

Either by expert intervention or using the Bisect algorithm one can obtain a set of intervals which essentially reduce the numeric attribute into a nominal attribute. Once this is done, any value occurring within the target can be classified as part of an interval. Notice that the discretization is similar to the calculation of priors in the preceding sections. Once an attribute of the master relation is discretized, the intervals can be used for all subsequent targets unless there is a significant change in the distribution of values in the source database. One can now define a random variable N_i which returns number pairs. Each possible value of N_i is the pair of boundaries for an interval. The probability of each interval returned by N_i is proportional to the conditional probability of a record being in $\mathbf{T}$ given the value of its n_i attribute is in the said interval.

Having defined the manner of feature selection, one can generate the population using selection closures that have a high probability of getting the correct answer. The initial population is composed of single-node trees. Each tree is therefore a basic single-predicate selection. A number of these equal to the size of the required initial population are generated. The size of the population stays constant during evolution. The actual number depends primarily on the application, size of the database, number of attributes and amount of processing power invested.

7.3 Evaluation of Individuals

Once the population is created, each individual must be evaluated in order to determine its fitness. The evaluation is done with respect to the precision and recall of each individual in approximating the target extension. At this point it might be worthwhile to define precision and recall in terms of the target extension $\mathbf{T}$ and the extension $\mathbf{Q_i}$ of an individual in the population:

$$Precision(\mathbf{Q}_i) = \frac{|\mathbf{Q}_i \bigcap \mathbf{T}|}{|\mathbf{Q}_i|} \tag{7.6}$$

$$Recall(\mathbf{Q}_i) = \frac{|\mathbf{Q}_i \bigcap \mathbf{T}|}{|\mathbf{T}|} \tag{7.7}$$

These two measures each give a different view of the performance. Precision essentially represents the performance in avoiding false positives, whereas recall is the measure of success in avoiding false negatives. Ultimately however, one needs a single measure to combine both performances in to a comparable fitness value. One such measure is the complement of *e-measure* [13] defined as a combination of the above two measures:

$$E^{'} = \frac{1+b^2}{\frac{b^2}{Recall(\mathbf{Q}_i)} + \frac{1}{Precision(\mathbf{Q}_i)}} \tag{7.8}$$

where b is a user defined parameter which allows one to favor recall over precision and vice versa. When $b = 1$, $E^{'}$ is identical to the harmonic mean of precision and recall. The advantage of using these measures is that they have well defined boundaries and behavior. The e-measure will always range between 0 and 1. It will tend to zero as $\mathbf{Q}_i$ and $\mathbf{T}$ grow disjoint and will tend to unity as they become identical.

7.3.1 Selection of Individuals

In genetic programming literature, the two most common methods of selecting individuals with higher fitness are *fitness-proportional* selection and *tournament* selection [50, 61]. In the former, individuals are selected with probability proportional to their fitnesses. In the latter, groups of 3 or 7 individuals are randomly selected and the individual with the highest fitness of each group becomes the selected individual. The main selection process repeats either method a number of times equal to the population size in order to obtain the parents for the new generation. Notice that several copies of the individuals with highest fitness are likely to be added to the parent pool whereas the lowest ranking individuals will not get a chance to breed at all.

Of these two methods, I shall use tournament selection with groups of 7. The choice in this case is neither easy nor absolute. However, the fact that tournament selection is easier to implement and easier to parallelize is a major advantage. Also, as with all local search methods genetic programming tends to converge as it proceeds. This is seen as diminished variation between individuals in the later generations. Proportional selection tends to overlook marginally better individuals in a population of equals. Tournament selection is more likely to capture small improvements.

7.3.2 Breeding of the Next Generation

Once a pool of parent individuals is prepared with selection, the next generation of individuals are created using three different operations. Each operation requires two individuals and in turn returns two new children. The next generation is not necessarily totally comprised of new individuals. A random portion of the parents may be injected into the new mix without any alteration. For example, in the experiments (cf. Sec. 8.4), 10 percent of the individuals of each generation were comprised of unmodified parents from the preceding generation. The remaining 90 percent were modified with a random probability of direct combination, mutation and crossover. The schematic representations of the recombination methods are given in Figure 7.3

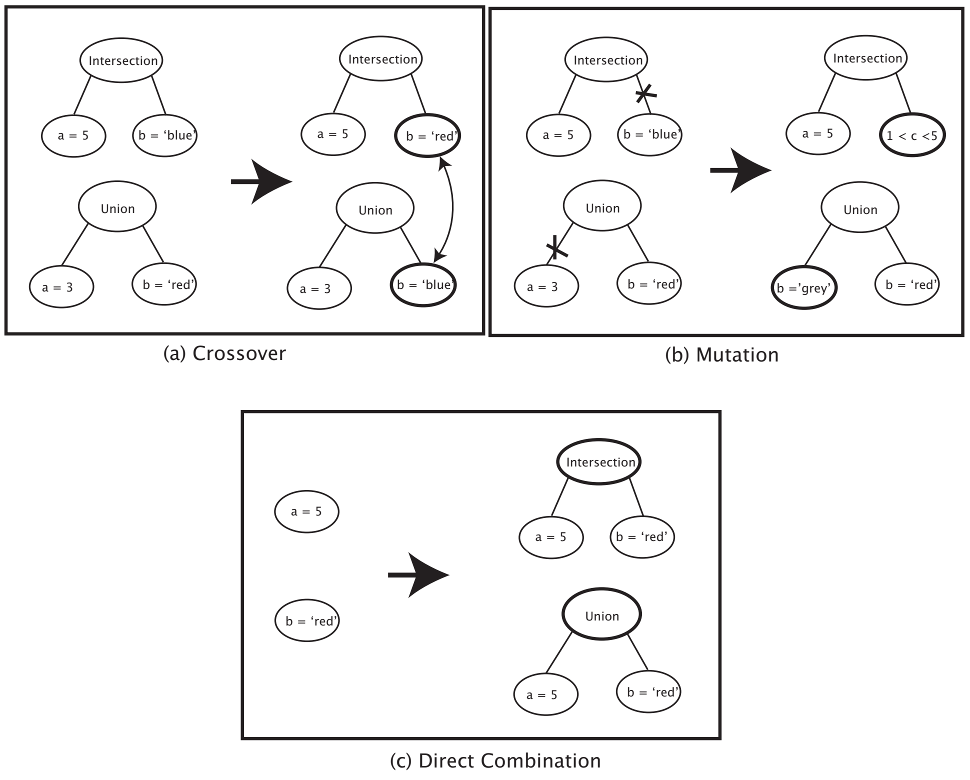


Figure 7.3: **Recombination Methods**
: (a) Crossover (b) Mutation (c) Direct Combination

Mutation

In this type of modification a pair of individuals from the parent pool are removed and modified separately. Remember that each individual is a tree with leaf nodes composed of selection closures and internal nodes composed of set operations. In mutation, for each one of the two individuals, a random node is selected and the tree is pruned at that node. A newly generated selection closure is added to the part of the tree that was cut. The resulting two new individuals are added to the new generation

Crossover

In crossover, a random node is removed from each of the parent individuals, along with any sub nodes connected to that node. The removed subtrees are switched such that the subtree removed from the first parent individual is connected to the point emptied by the subtree removed from the other individual and vice versa. The resulting trees are added to the next generation.

Direct Combination

In direct combination, the two parent individuals are copied to obtain two pairs. Each pair is connected by a new root node. This root node is selected as a random set operation from the three possible operations defined in Section 3.2. The resulting larger individuals are added to the new generation.

The new generation being thus generated, the cycle is repeated as shown in Figure 7.1. The cycle repeats until one individual attains a fitness above the threshold or a given number of generations pass without any improvement in the best fitness so far.

7.3.3 Optimizations

Several optimizations are done in implementation, primarily to increase the speed of the system and also to improve the quality of the results. The main optimization is the caching of intermediate answers. Since the leaves of each query tree are selection operators, these are executed as simple selections on the master relation **R** and the resulting tuples are stored in memory as efficient bit vectors. Hence, the subsequent set of operations done as a result of combining the leaves through intermediate nodes up to the root is done in memory and on CPU time. This is a major improvement over sending each query directly to the DBMS for processing. The DBMS and disk access steps are shown to be the biggest bottlenecks in the next section. Therefore, it makes sense to remove as much of the processing from the DBMS to the CPU. The intermediate nodes, likewise also cache their results and return these directly unless their descendant nodes have changed due to mutation or crossover.

The second optimization involves building the individuals up from the smallest possible size and grow them gradually as the generations advance. This ensures that it will be more likely to get the most concise intensional answer for a given target. Concordantly, the zeroth generation is composed solely of single nodes which are subsequently grown into more complex and larger queries primarily through the direct combination method. Therefore, the probability of the system selecting a direct combination during breeding is given by a function inversely proportional to the size of the trees that are to be combined. For example, for individual of size 1 (i.e., single node queries) the system will use direct combination. For trees of size 2 (i.e., two leaves combined by a single set operation) the probability of using direct combination is 0.5, the probabilities of crossover and mutation are 0.25 each. For trees of size 3 the respective probabilities are 0.33, 0.33, 0.33 and so forth.

7.4 Summary

This chapter describes a method of re-encapsulating the extensions of candidate consolidation with new intensional expressions (i.e., those different from the expressions generated originally by the techniques described in Chapters 4 and 6).

The method is based on genetic programming and evolves populations of candidate query expressions. The fitness constraint driving the evolution is geared towards increased accuracy in generating the desired extension as well as being concise in terms of expression size. The initial population is not random and is seeded with individuals more likely to contain selection predicates with comparisons to values that are more selective towards the target extension, using the naïve Bayes model.

Chapter 8: Experimental Results

In this chapter I will describe the experimental validation performed on the methods discussed thus far, in terms of both the accuracy of the methods and the time requirements. I will, for each suite of experiments, also discuss the variation of accuracy and time performance with respect to factors relevant to each experiment.

Section 8.1 will describe experiments performed on the conditional random field labelling approach for query sequences. Section 8.2 will explain the experimental analysis of the minimal consolidation generation and ranking method defined in Chapter 4, with particular attention to the differences in accuracy and performance of the exact and sampling variants of the method. Section 8.3 will validate the association analysis approach for finding likely global selection constraints to the join plans. Finally, Section 8.4 will analyze the experimental results of the genetic programming based re-encapsulation approach.

8.1 Sequence Labeling and Segmentation

I performed a suite of experiments on the CRF labelling approach in order to validate its accuracy and examine its behavior. Two widely publicly available benchmark databases,TPC-H [89] and Mondial [90], were used for their schemata. Of these, TPC-H is the simulated database of a business, including supplier and customer information, invoices, orders and so forth. It has 8 tables. These 8 tables, for the purposes of this experiment, are treated as different data sources. Furthermore, the larger tables have been vertically sliced into smaller parts, each one as a different source. This allows one to simulate the vertically distributed data in a virtual database environment. Therefore, with the subdivisions the TPC-H test scenario simulates 17 different sources in a global virtual database schema.

Table 8.1: **Sizes of Generated Query Sequences**

Sequence Length	Mondial	TPC-H
Minimum	1	1
Median	10	7
Maximum	36	23

The second test platform, the Mondial database, has a larger schema and is populated with real-world geopolitical data from several resources (e.g., the CIA World Fact Book). Originally it has 23 tables, the larger of which were likewise vertically subdivided in order to obtain 30 simulated data sources.

Some parts of the subdivided tables were given binding limitations (e.g., the Countries table in Mondial can only be searched by setting name of the country)

In order to create training and test data for the CRF, I generated 500 query sequences for each. This was done by first creating a universal relation for each database. Random views of these universal relations were then set as targets and a query plan (i.e., a query sequence) was generated for each target, along with the labels for each of the components.

Due to the differences of the schemas and the random nature of the target selection, the lengths of the query sequences are distributed over a range of values. Table 8.1 summarizes these distributions.

The 500 sequences where randomly concatenated into groups of 10, each group becoming a simulated query log for one user. The time stamps on the queries were simulated with a Poisson process, with the rate, $\lambda\tau$, being simulated for different inter-query gaps. The rate parameters, λ, were set in a ratio of 1:2:4, for gaps between component queries of composite subgoals, gaps between goals, and gaps between goals, respectively. The time constant of the process, τ, was the kept the same throughout.

The CRF was trained with 49 groups and tested with the remaining one using leave-one-out cross-validation. Therefore, each database scenario was run 50 different times with a different training partition of 1:49.

Three different CRF scenarios were experimented with. The CRF experiment is the basic approach with all the possible features. The PrunedCRF experiment is the CRF scenario trained with gain-based pruning of attributes during training. Finally, the ContentCRF scenario is a control test where extra features were added based on the actual content of the databases. The training and testing programs were implemented in Java and the experiments were run on a 2.2 MHz Athlon 64 system with 1 GB of memory operating under FreeBSD 6. Each cross-validation experiment (i.e., 50 runs) was completed in times ranging from 1 to 32 hours, on this platform, depending on the number of features. The training is by far the larger time-consumer while labeling of the average test group took around 1.5 seconds. Figure 8.1 shows the progression of the optimization of the log-likelihood for the various experiments with respect to amount of training time. Table 8.1 shows the number of active features per clique for each scenario.

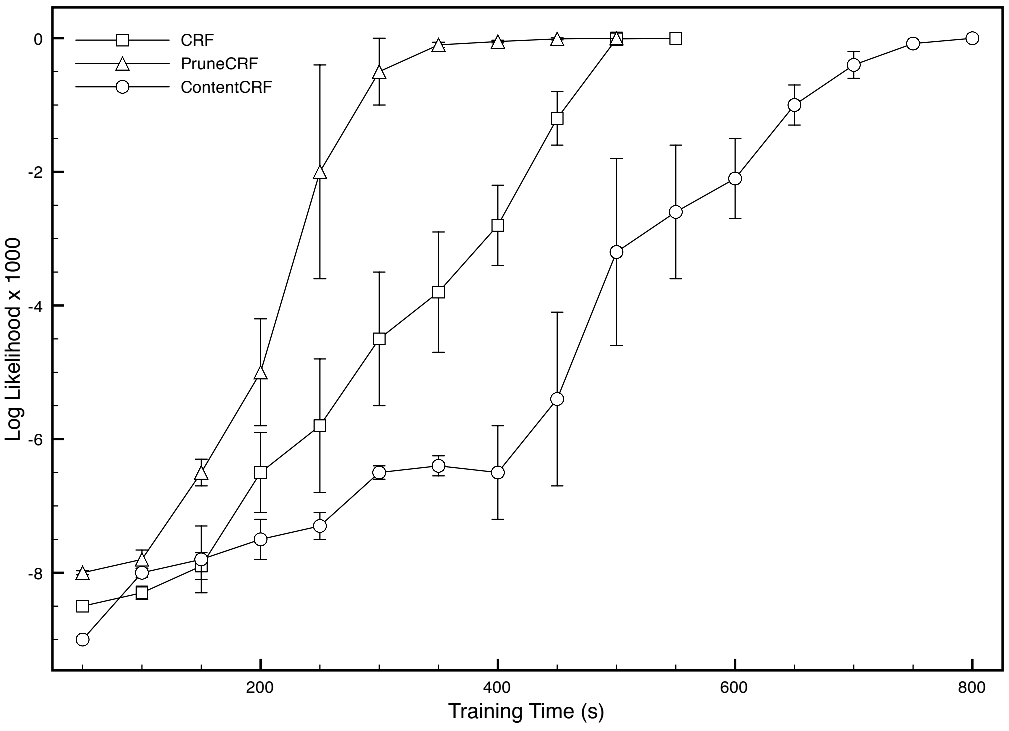


Figure 8.1: **Log-Likelihood vs. Training Time**

The training was done using gradient ascent, particularly the L-BFGS [60] method. To avoid overfitting, the likelihood function was regularized with a Gaussion prior with variance

Table 8.2: **Number of Features for the CRF Experiments**

Method	Mondial	TPC-H
CRF	12,278	12,278
PrunedCRF	1,065	983
ContentCRF	150,271	132,055

5. After each round of training with 49 groups the remaining test group was labelled and the labels were compared to the true labels which were decided during the generation of the query sequences. The accuracy of a test is defined as the percentage of labels that are the same with the correct labels.

The robustness of the CRF method to noise was also tested during these experiments. For each cross-validation run, after the training of the CRF, the test sequence was tested for accuracy initially as it was. Subsequently the test cases were each "corrupted" by inverting the values of random observation features in the test sequence. For example, if the one query was actually joinable with another, the noise induction might flip the value showing the queries as non-joinable. By this method, I aim to simulate user and observation errors. The tests were run at noise levels of 5, 10, and 15%, relating to that fraction of observation features being corrupted. Table 8.1 shows the results.

Table 8.1 presents the average accuracy of each cross-validation run, per scenario and per noise level.

The results are encouraging in that even at relatively high noise levels the feature pruned CRF and the content enriched CRF are still reasonably accurate (on average above 95% accuracy at 1-in-20 error in observation). The basic CRF seems to suffer immediately from the noise. This is undoubtedly due to the tendency of the unpruned features to over-fit the training data. Therefore, it is clear that pruned feature spaces have the additional benefit of lower generalization errors, even in the case of noisy input. The content enriched feature set tends to perform best in high noise situations. However, this slight gain in accuracy arguably does not offset the fact that the feature rich model is larger, slower and less flexible.

As a subset of the preceding analysis, I have also investigated the boundary detection

Table 8.3: **Labeling Accuracy Results**

		Test Accuracy (% Correct)	
Noise Level	Method	TPC-H	Mondial
0 %	CRF	94.6	91.2
	PrunedCRF	96.3	95.4
	ContentCRF	99.2	98.4
5 %	CRF	86.3	88.1
	PrunedCRF	95.2	92.3
	ContentCRF	96.7	94.1
10 %	CRF	73.2	70.6
	PrunedCRF	82.8	79.5
	ContentCRF	83.2	81.4
15 %	CRF	60.3	65.1
	PrunedCRF	68.6	69.3
	ContentCRF	73.6	75.9

capabilities of each model. This is purely to test whether the system segments the sequences correctly, regardless of the correct labeling of these segments. I test the accuracy of this in terms of precision and recall. The *precision* of boundary detection is defined as the ratio of correct segment boundaries the recognized to the total number of boundaries it detected. In other words, it is the inverse ratio of false positives. In contrast, the *recall* of boundary detection is the ratio of correctly detected boundaries to the number of actual boundaries that existed in the sequence. Likewise, this is an inverse measure of false negatives. One commonly used combination of precision and recall into a single accuracy measure is the F1 measure:

$$F1 = \frac{2PR}{P + R}$$

where P is precision and R is recall. The results analogous to the labeling results are given in Table 8.1. The segmentation behavior of the CRF is much better than the more complicated task of labeling, with very high accuracies up to very high noise levels. As with the labeling task, the basic, unpruned CRF suffers as noise increases, for the same reasons. In the segmenting task the difference in accuracy for the content enriched model is even less

Table 8.4: **Segmentation Accuracy Results**

		Test Accuracy (F-1)	
Noise Level	Method	TPC-H	Mondial
0 %	CRF	99.2	99.7
	PrunedCRF	99.9	99.9
	ContentCRF	99.8	99.9
5 %	CRF	98.4	97.3
	PrunedCRF	98.7	98.2
	ContentCRF	99.7	99.9
10 %	CRF	97.1	97.1
	PrunedCRF	98.2	98.8
	ContentCRF	99.1	99.3
15 %	CRF	91.2	90.3
	PrunedCRF	93.8	96.2
	ContentCRF	96.4	97.7

pronounced, making it even less useful from a cost-reward perspective.

8.2 Generation and Ranking of Minimal Consolidations

I have implemented the ideas explained in Chapter 4 in order to validate the approach and to better understand the behavior of individual variantes. I analyze the results in terms of both accuracy and performance along with their variations with respect to different factors.

All experimentation was done on a 2.33 GHz x86 workstation with 2 GB of memory and running OS X 10.4.9. While the processor used was a dual-core model, all processing was locked to a single core. The DBMS used was MySQL 5.0 and, when used, the server process of the DBMS was operated on the other core. The implementation was done in Ruby for most of the code. There are several exceptions, such as the bitmaps used for rapid item counting and the DBMS adapter which were natively compiled extensions written in C for the purposes of speed and memory optimization. The absolute time performance will obviously be highly variable with respect to factors such as the hardware used, parallelization, the use of native code and software optimizations. I provide the test system details merely so that when absolute time values are reported (e.g., in seconds) later on, the reader may have an idea of the machinery involved. All such time values are thread CPU times as opposed to wall-clock time.

8.2.1 Building of Candidate Joins

The query consolidation starts with the forming a join graph for joining the relations that embody the results of each local query. One would like to see how well the discovery and ranking of possible joins works. To this end I use the Mondial database [90].

To see the way experimentation is conducted, I will go through an example run with four relations to be joined. In this example the input relations are Country, Organization, IsMember and City. The system initially selects a set of attributes from each of these relations that are suitable for use as join attributes. In the experiments the class of join attributes are defined as either text attributes of less than 20 characters or integer attributes. So the attributes considered for each of the relations in the example are:

Country Name, Code, Capital

City Name, Country

IsMember Country, Organization, Type

Organization Abbreviation, City, Country, Province

The primary keys of each relation are underlined for the benefit of the reader. The joining system does not have any information as to which attribute is key or the relationship between relations.

After the relations are input, the system checks the pairwise overlap between attributes of pairs of relations in order to find out which attributes are compatible (cf. Sec. 4.1 Eqn. 4.1).

This information is used by the program to build a join graph. The edges in the graph are weighted with respect to the information gain of each attribute on the target relation (c.f. Eqns. 4.3 thru 4.6). Figure 8.2 shows the join graph generated for the example case.

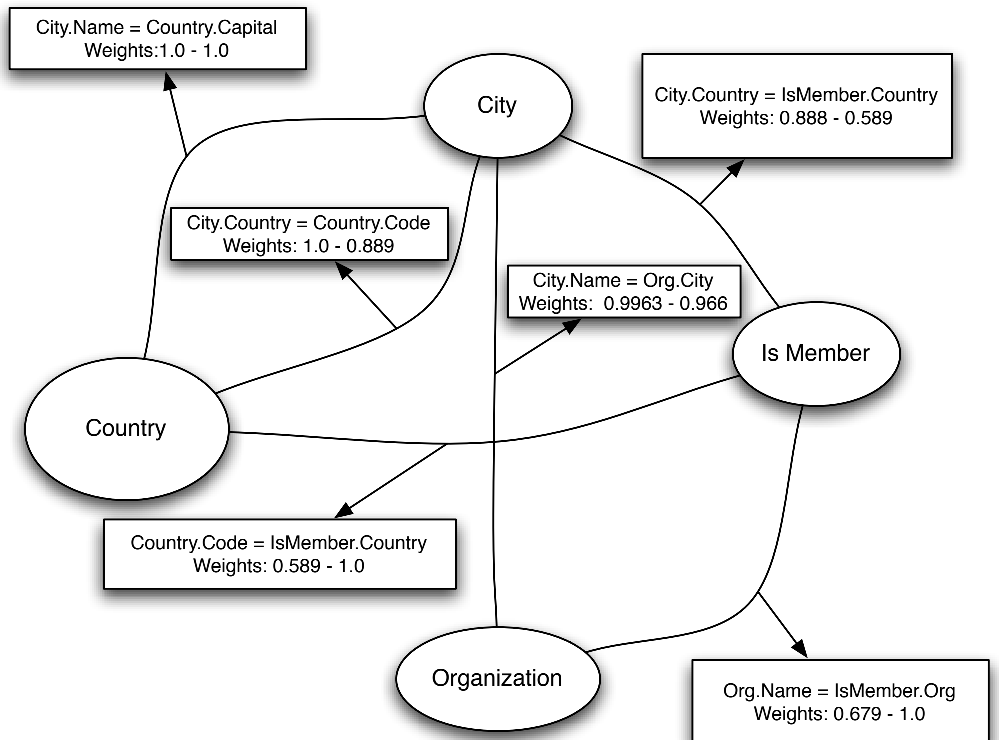


Figure 8.2: **Weighted Join Graph**

Each node in the graph is a relation and the edges each denote one possible join. The edges are all labeled with callouts detailing the join attributes involved in the joins. Each callout also contains the information gain of each side of the join.

The system then generates the top-k spanning trees of the join graph using the algorithm described in [46]. As mentioned before, each edge is weighted with the maximum of the pair of information gains. Ties are resolved by prioritizing edges with higher second weight. For example, in Figure 8.2 the edge labeled "City.Name = Country.Capital" has information gains {1.0,1.0}. The maximum is 1.0 so the weight of that edge is 1.0 (i.e., maximum of 1.0 and 1.0). However, "City.Country = Country.Code" has a gain vector of {1.0,0.889} leading, again, to a weight of 1.0. When building a spanning tree, these two edges are mutually exclusive since they form a cycle. Therefore, each spanning tree having the former edge has an alternative spanning tree that uses the latter (e.g., Fig.8.3 parts (a) and (b)). All else being equal, both spanning trees would have the same total weight. However, spanning tree having the former edge higher than the latter is ranked higher since the minimum of its information gain vector is greater (i.e., $1.0 > 0.889$).

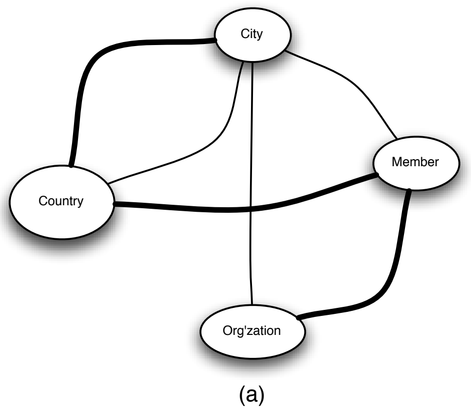
Figure 8.3 shows the first six highest ranked join plans out of 13 possible for the example join graph. The natural language explanations of these joins are:

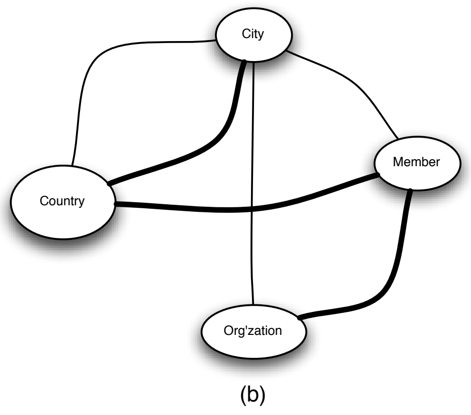
Join (a): countries each with information on its capital city and information on organizations it is a member to.

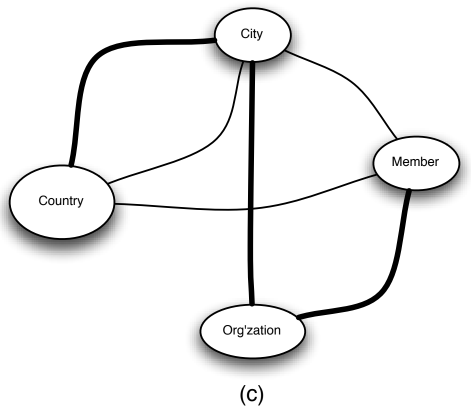
Join (b): cities each with information on the country it belongs to and the organizations that country is member to.

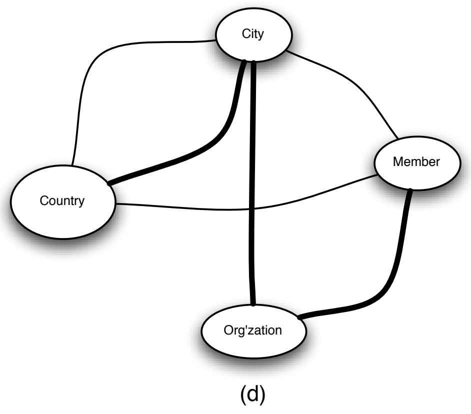
Join (c): organizations which are headquartered in a capital city, information on the host country along with country codes of members.

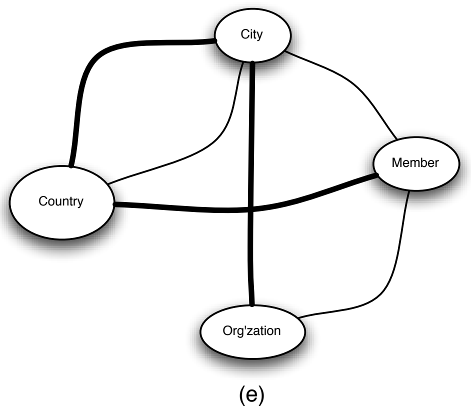
Join (d): organizations with information on the host city and country along with country codes of members.











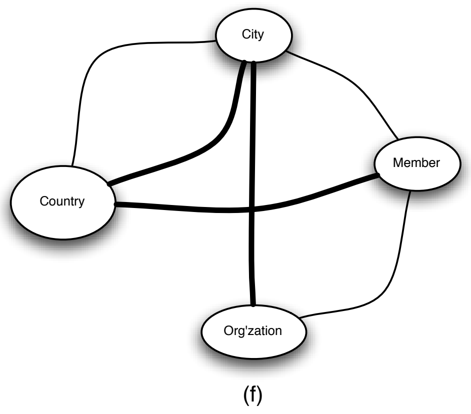


Figure 8.3: **Span Tree Ranking**
Ranking is in letter order. Trees a and b have the same weight (3.0). Trees c, d, e and f have the same weight (2.96)

Join (e): countries with any organizations they host in their capitals along with abbreviations of organizations they belong to.

Join (f): countries with any organizations they host in any of their cities along with abbreviations of organizations they belong to.

One can see that each of these joins are meaningful in some way, hence the requirement of ranking a set of possible explanations rather than coming up with a single answer. However, due to the ranking scheme, joins that are simpler to express and thus more probable are ranked higher than others.

I have run a number of such trials with different portions of the Mondial schema and varying number of relations to be joined in each case. A totally objective way of measuring accuracy does not exist, given that a number of relations can be joined multiple ways and still make sense. I therefore adopt the following method.

The whole schema of Mondial contains 28 relations, ranging from two to six attributes each. Each iteration of the experiment takes a random subgraph of n relations of the whole schema. The key relationships are known from the schema so a target join is created from the this path between the n relations. For example, in the example above, the target join selected would be:

$$Country \bowtie_{Cap.=Name} City \bowtie_{Code=Country} IsMember \bowtie_{Org.=Abbr.} Organization \qquad (8.1)$$

This target join is stored and the relations are given, as is, to the system. The system creates a join plan and ranks possible joins. I then search for the target join within the ranking. The rank at which the target join is found determines the success of the approach.

The system was run on subgraphs ranging from 3 to 8 relations with 20 random subgraphs for each complexity level. Table 8.5 summarizes the results. The algoritm, as expected, performs very well in simple cases (e.g., joins involving 3,4 or 5 relations) where

Table 8.5: **Join Inference Accuracy with respect to Join Complexity**

Number of Relations	Number of Joins				Ranking of Target			
	Mean	Median	Min.	Max.	Mean	Median	Highest	Lowest
3	3.65	2.5	1	12	1.45	1	1	6
4	5.65	5	1	13	1.7	1	1	9
5	12.4	5.5	1	88	2.65	1	1	12
6	33.8	18.5	1	148	4.05	2	1	17
7	55.95	21	2	285	7.95	5	3	29
8	176.5	53.5	5	1089	20.8	10	2	109

the target join is ranked on top more than half of the time. Even in joins with 8 relations, in more than half of the runs, the target join is within first 10 ranked explanations.

8.2.2 Comparison of Exact and Approximate Join Inference

The TPC-H [89] benchmark database was used for testing both the time performance of the exact method as well as testing the accuracy and performance of the approximate (sampling) variant.

The exact method runs off of a database management system (MySQL 5.0). The approximate method starts off with randomly sampling a small subset of each relation into memory and subsequently works totally in memory. The random sampling requires one scan of the table and thus the database access (i.e., disk access) for the approximate method is only linearly dependent on the size of the original relation. On the other hand, the exact method uses the DBMS extensively while calculating domain overlap and subsequently during testing the plausibility of edges (i.e., while weighting the possible join candidates).

Performance

The exact method was tested for performance by analyzing a possible join of a table[1] from the TPC-H database with itself. This is a worst-case test of the method since when a table is joined with an identical copy, every column matches at least one other column in the counterpart. I have tested with various horizontal and vertical subsets of the table,

[1]the LineItem table, which is the largest, both in the number of rows and columns

including subsets with 100, 500, 1000, 1500, 2000, 2500 and 3000 rows each with 2, 3 and 4 joinable attributes. Each of the 21 scenarios were run 20 times each to average the time required. The DBMS query cache was flushed after each run to avoid erroneously fast performance. The times required to generate the weighted join graph (in this case a bipartite graph since the test invloves two relations) between the two tables are given in Figure 8.4. Each point represents the mean run time of the 20 runs at each configuration and the error bars represent standard deviation.

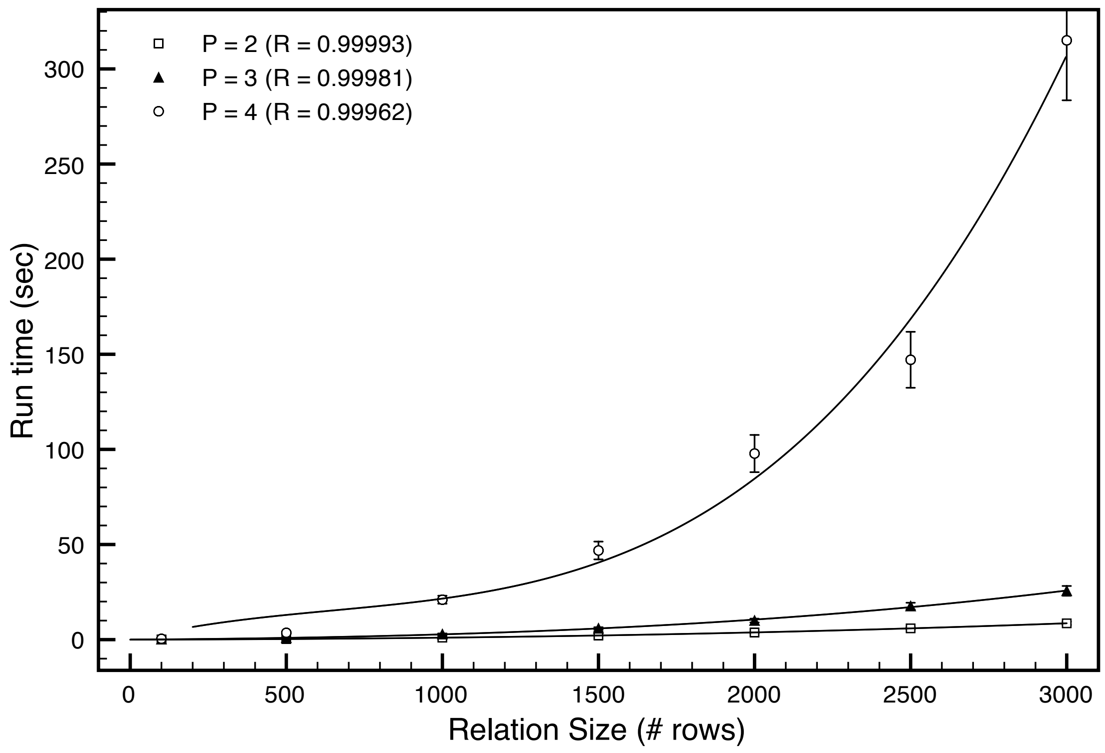


Figure 8.4: **Running Times of the Exact Method**

It is clear from the figure that the exact method is too costly to use in any situation where either the arity or the size of the relations involved is considerable. Recall from Sec. 4.2.4 that the growth with respect to relation size is weaker (i.e., N log N). However, the growth with respect to the average arity (i.e., P) is cubic in the worst case. Notice that, since I am experimenting with join plans involving two identical tables, the number of relations (K) is fixed at two and recall that the exact method will present quadratic growth with increasing numbers of relations involved.

Alternatively, the approximate method is several orders of magnitude faster and scales much more favorably with respect to all these parameters. In order to test this, I have performed the same experiment with the approximate method albeit with different sizes. The original table was again LineItem, with a total of 100,000 rows. I have run the experiments with sampling rates of 0.001, 0.01, 0.05, 0.1, 0.2, 0.3, 0.4 and 0.5[2]. Each point was run a total of 100 times. Figure 8.5 gives the mean runtimes for each scenario. The data points were fit with functions of the form $a + bx\log(cx)$ in order to verify the expected scaling with respect to sample size. These fits are shown as well in Fig. 8.5.

An approximate method run starts by a pseudo-random sampling of the original table for the required sample size. A the MT19937 variant of the Mersenne Twister PRNG is used and the generator is seeded with the system timer at the beginning of each run. The rest of the operation is conducted completely in memory. This includes finding domain matches through the induction of simple pattern matching DFAs and the calculation of the conditional entropy of the relation given each attribute. For the purposes of Figure 8.5, the effective arity (P) of the relation was 6. Although the relation has 16 columns, only 6 of them satisfy the requirement that equi-join attributes be either integers or short strings. Of these candidates, some were removed in order to test the effect of the arity. Each level is shown as a different curve in Fig. 8.5. Notice that, with even large sample sizes (e.g., 10000 rows) the operation is quick enough to be completed within a second on modest hardware. In a production setting this may be cut down considerably with the use of parallel processing. It is possible to do the calculations of each attribute separately from each other on multiple processors giving the operation a virtually linear speedup hence allowing online operation. A sample of 10000 rows for this example relates to a sample rate of 0.1 on the original relation which has 100000 rows.

[2]Corresponding to sample sizes of 100, 1000, 5000, 10000, 20000, 30000, 40000 and 50000 rows, respectively

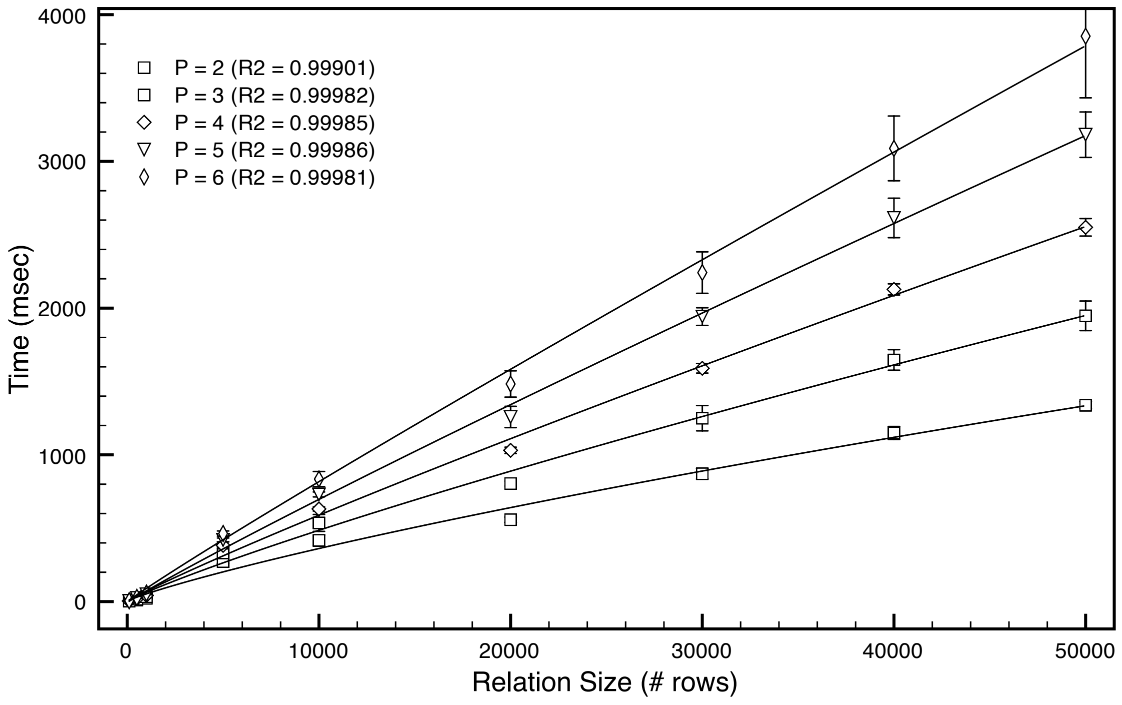


Figure 8.5: **Running Times of the Sampling Method**

The scaling of the computation time with respect to the number of candidate attributes was also further investigated. The approximate method is expected to scale quadratically with the number of possible join attributes. None of the tables of the TPC-H benchmark database contain more than 16 attributes. In order to test the worst case scaling of the method, a synthetic table of 50000 tuples and 100 attributes was generated to simulate the worst case. The table is such that every attribute is a key and each is domain-wise compatible with every other attribute in the table. Hence the method is forced to consider a complete join graph in a scenario where this test table is joined with itself.

Figure 8.6 shows the results of this simulation. The simulation was run once for various projections of different arity. The resulting data was fitted with a quadratic function, which is shown as well. As expected, the approximate method scales quadratically with respect to the number of candidate join attributes. Notice however, that the quadratic nature is not very pronounced for arities less than 10. Therefore, for relations of arity less than

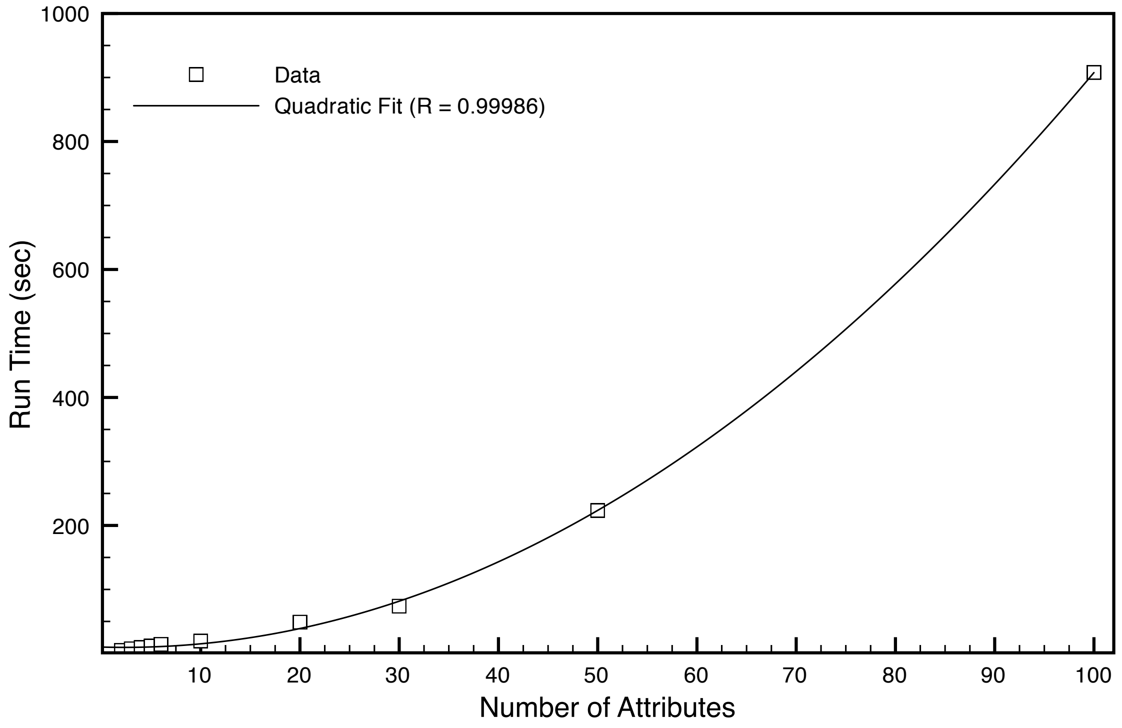


Figure 8.6: **Running Times of the Sampling Method with Respect to Number of Join Attributes**

10, the performance should not be affected. Due to database normalization and the human attention span[3] it is uncommon for relations to have many attributes and the join candidate attributes will be fewer still.

Accuracy

I test the accuracy of the approximate method taking the exact method as a standard. The exact method takes into consideration all the tuples involved and tests pairs of relations against each other while scoring the plausability of each possible join. The exact method also is able to find the exact domain overlap by actually intersecting the complete tables with each other. The approximate method aims to estimate the same by using only a fraction of the tuples from each relation. Furthermore, the calculation on each relation is self-contained.

[3]The human cognitive channel limits the number of items that can be retained by the average user at about 7 [68]

While this is the reason the approximate method scales favorably it also requires that the degree to which it correctly approximates the full calculation be investigated.

The weights assigned by the approximate method to each edge in the join graph consist of two distinct terms. The first is the extent to which the domains of the attributes at each end of the edge match each other. This is determined by classifying the domain of one sampled attribute with a DFA trained on the domain of the other. The second factor is the extent to which either attribute serves as a key on its own relation. This determination is made by calculating the prior entropy of the sampled relation and the conditional entropy of same given the attribute and calculating the relative change (i.e., finding the information gain). Since the idea is to estimate this statistic based on a sample of the original population (i.e., the whole relation) it makes sense that the accuracy of the method depends primarily on the relative size of the sample.

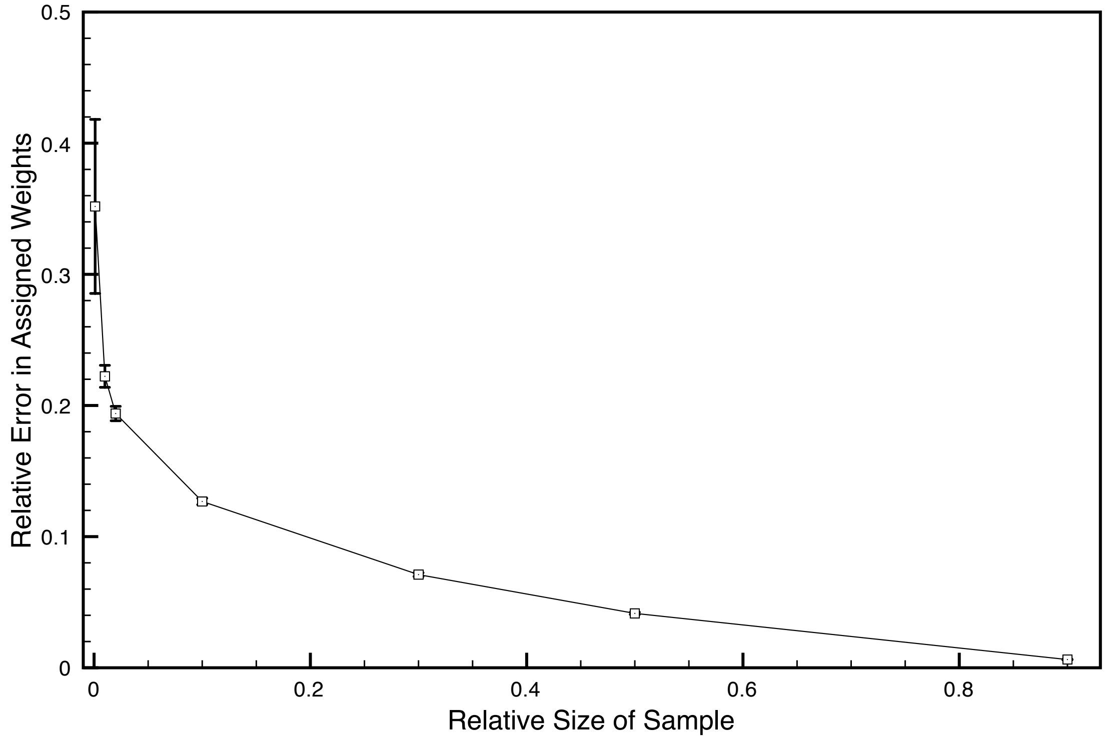


Figure 8.7: **Error in Weights Assigned by the Sampling Method with Respect to the Exact Method**

In order to investigate the effects of varying sample size on accuracy, the experiment is performed in a bottom up fashion, starting with the error in individual edge weights and working up to the ranking of join plans. Figure 8.7 shows the error in weighting of edges with varying sampling rates. The weighting error for any edge $< A_1, A_2 >$ is defined as:

$$E(A_1, A_2) = \frac{|w_{exact}(A_1, A_2) - w_{approx}(A_1, A_2)|}{w_{exact}(A_1, A_2)}$$

Each point in the figure is thus the mean weighting error for all edges in pairwise join inferences between tables in the TPC-H database. The error bars represent the standard deviation in the weighting error. This variance is negligible in cases where the sample size is more than 5% of the total relation. Hence, as long as the sample size is within an order of magnitude of the original relation, the behavior of the approximate method is stable enough to be considered to be deterministic.

Figure 8.8 shows the effects of the weighting error on ranking. The LineItem table in the TPC-H was experimented on in order to be able to consistently compare ranking of the exact method and the approximate variant. The experiments were done on a pair of tables. Since the method is locally optimized, if the links between every pair of tables is weighted correctly the whole join graph is assumed to be correctly weighted. Therefore, without loss of generality the focus of the experimentation is pairwise weighting between tables. In this case, there are 12 possible edges in the join graph and the original relation has 100000 rows. The exact method was run once to find the benchmark ranking. Subsequently, the approximate method was run 20 times each at various sample sizes and made to rank the same 12 edges. The figure shows the mean ranking error as the sample size increases. The error bars represent the standard deviation. Even at samples of 70 rows (i.e., a sample less than 1% of the original.), the ranking of the approximate method stabilizes at a ranking error of 2 (i.e., each edge is ranked on average two places away from their position in the benchmark ranking). As the sample size becomes large enough to comprise a fraction of the original relation (e.g., 20% of the whole) the ranking of the approximate method converges to that of the exact method.

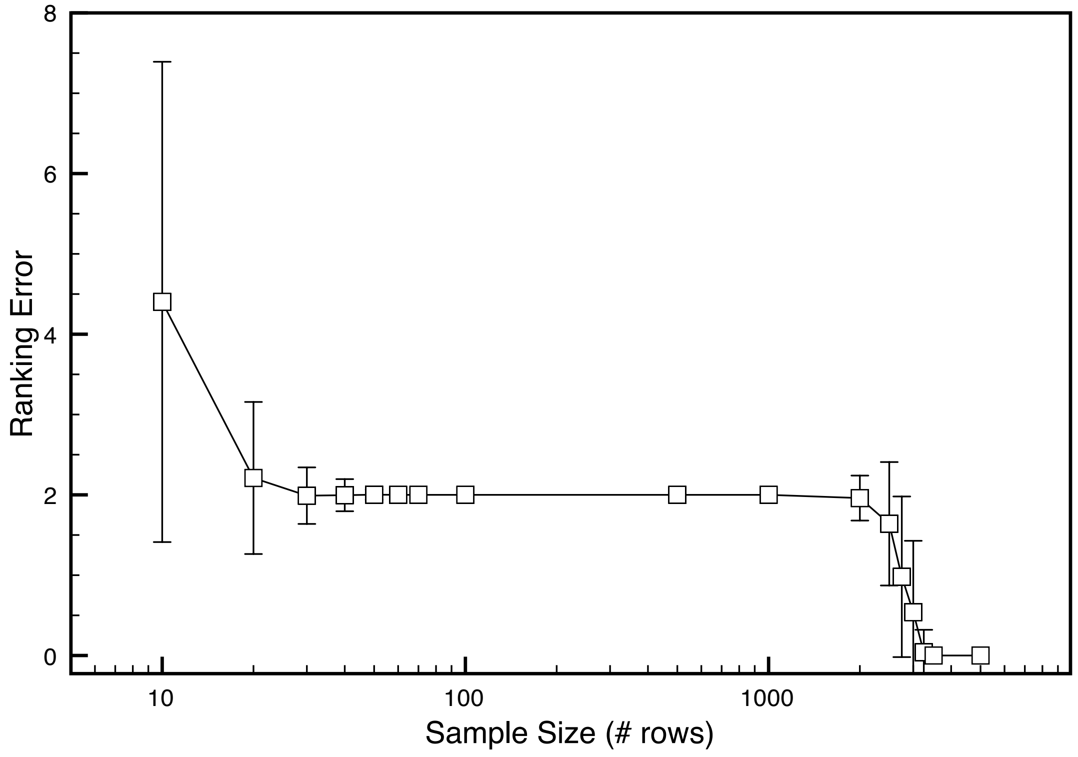


Figure 8.8: **Average Error in Candidate Join Rank of the Sampling Method with respect to the Exact Method**

Therefore, for normally or uniformly distributed tables, the approximate method is able to pick out distinctly better join plans and rank them at the top, even when the sample taken is one hundredth of the total population. However, it is difficult for the method to differentiate the more 'mediocre' join plans without more information. This is clearly seen in 8.8 the initial inflection at around a sample size of 50 represents the point where the most promising joins have settled in the top ranks. This is where the most plausible join is discovered. As the sample size, and hence the information available to the method, increases even the lower rankings converge to the correct ranking. This is seen as the second inflection in the figure at around a sample size of 2000 (i.e., a 20% sample). Figure 8.9 further illustrates this effect. At the former inflection, the top join plan has stabilized and is thereon consistently found as the top join. After the second inflection, all rankings are consistently the same with the benchmark ranking (denoted as the percentage of 'perfect rankings' in the figure). Therefore, the approximate method can consistently find the top ranked join plans at 1% sample size and is virtually the same as the exact method for sample sizes above 20%.

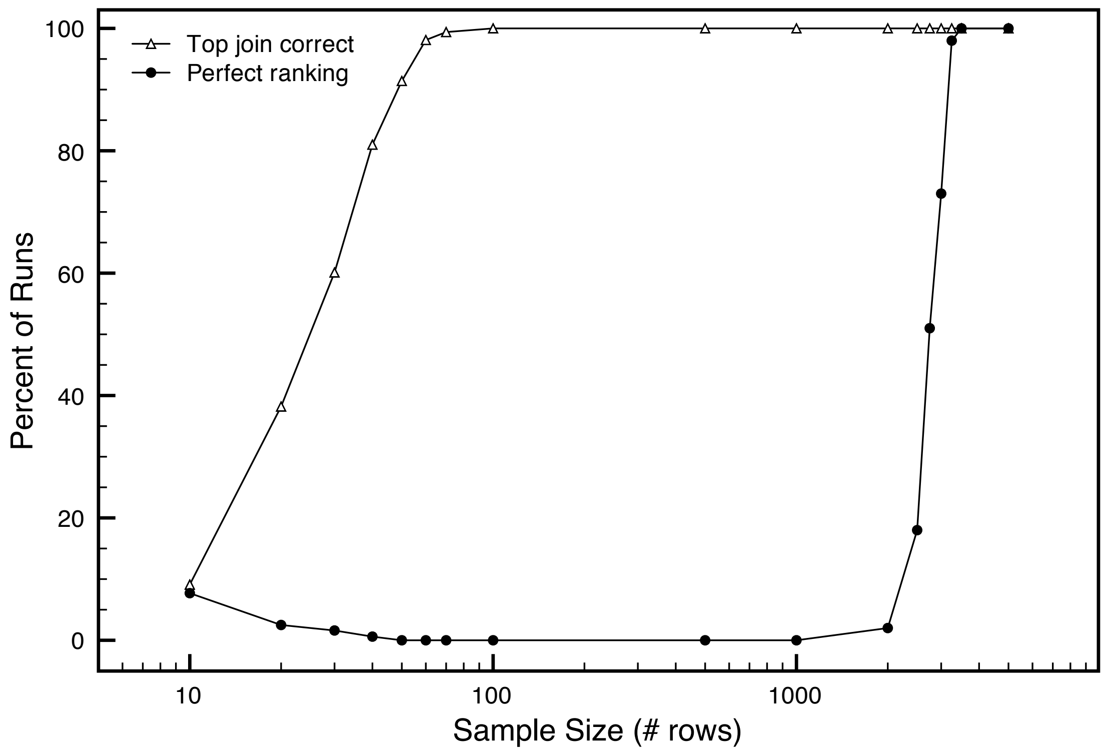


Figure 8.9: **Accuracy of the Total Ranking of the Sampling Method with Respect to the Exact Method**

8.3 Association of Projections and Constraints

In order to analyze the behavior and performance of the association analysis method described in section 6.2 I have implemented the MAFIA algorithm descirbed in [20] in Ruby with the added optimizations discussed in said section.

The system works by creating and storing the occurrence of each single item in the training set of queries by building a compact and fast bit vector stored in memory. Itemsets larger than 1 item thus do not require one to rescan the training set. Any occurences of a k-itemset can be obtained by intersecting the bit vector of its parent (i.e., a (k-1)-itemset) with the bitvector of the item to be added to the set. These bitvector classes were implemented in C and compiled natively in order to preserve memory and prevent an unnecessary performance drag. The remainder of the system, including tree-traversal and mathematical calculations are pure Ruby.

The training set used was obtained from the Sloan Digital Sky Survey [87] and is a set of SQL queries on the SDSS SkyServer database. The SkyServer is an online database housing the most extensive astronomical survey to date. It contains visual, geometric and spectrographic information on millions of astronomical bodies such as stars, galaxies, clusters &c. This set was chosen because, due to the nature of the subject, the queries contain an abundance of comparisons between attributes such as declination, right ascension, velocities and colors of objects.

These queries were parsed and the projected attribute names and comparisons in each were into a flat file containing ordered pairs. Each pair is organized as {Set of Projected Attributes, Set of Comparisons}. During parsing any nested queries were extracted and added as separate queries.

The association analysis was run using three-fold cross-validation. After training, for each of the rules derived, the test set was filtered for pairs having the whole antecedent of the rule in their projection set. The filtered set was then counted for pairs having the consequent of the rule in their set of comparisons. The proportion of the pairs having both the required projection and the comparison to those having only the projection is the confidence of the rule with respect to the test set.

I measure the prediction error of a rule i as follows:

$$E_i^{Prediction} = (C_i^{Test} - C_i^{Training})^2 \tag{8.2}$$

The average error of a cross-validation fold is defined as the square root of the mean of the errors of all rules in that fold. The average error is the mean error over all three folds of cross-validation. Table 8.6 presents the experimental results for training done at different levels of minimum support.

Table 8.6: **Prediction Errors with Respect to Support and Confidence Thresholds**

Support Threshold	**Confidence Threshold**	**Number of Rules Found**	**Prediction Error**	
			Mean	**St. Dev.**
0.05	0	11	0.08	0.03
	0.5	7	0.08	0.035
0.1	0	4	0.09	0.06
	0.5	2	0.05	0.04
0.2	0	2	0.03	0.02
	0.5	1	0.04	0.03

The results confirm that the method does not particularly over- or under-estimate the confidence of a rule. The decision to rank rules, and the comparisons they suggest, with respect to their confidences is therefore justified. The error rates do not seem to be sensitive to the minimum support threshold. Although higher thresholds seem to have the effect of lowering the mean error rate, the variance between runs of cross-validation within the same group are higher than the difference in mean errors. This has been indepedently verified with ANOVA. The p-value of the F-test turns out to be 0.454, so one can say that the error rate is not dependent on the support threshold.

Likewise, there seems to be no relationship between the confidence of a rule and its prediction error. This is shown in the table where the error statistics are repeated for the subset of rules above %50 confidence, in each case.

Hence, one can then say that while training, the support threshold should be set as low as possible while still keeping the resulting number of training samples statistically significant, as explained in Sec. 6.2.

That being said, decreasing the support threshold will cause a significant time cost during training. As the support threshold drops, the number of candidate itemsets that can be pruned out also drops dramatically. While this may lead to the discovery of more rules, the time required to train the system will rise exponentially. This is shown in Figure 8.10 part (a), where the run times of the analysis was recorded for various support thresholds.

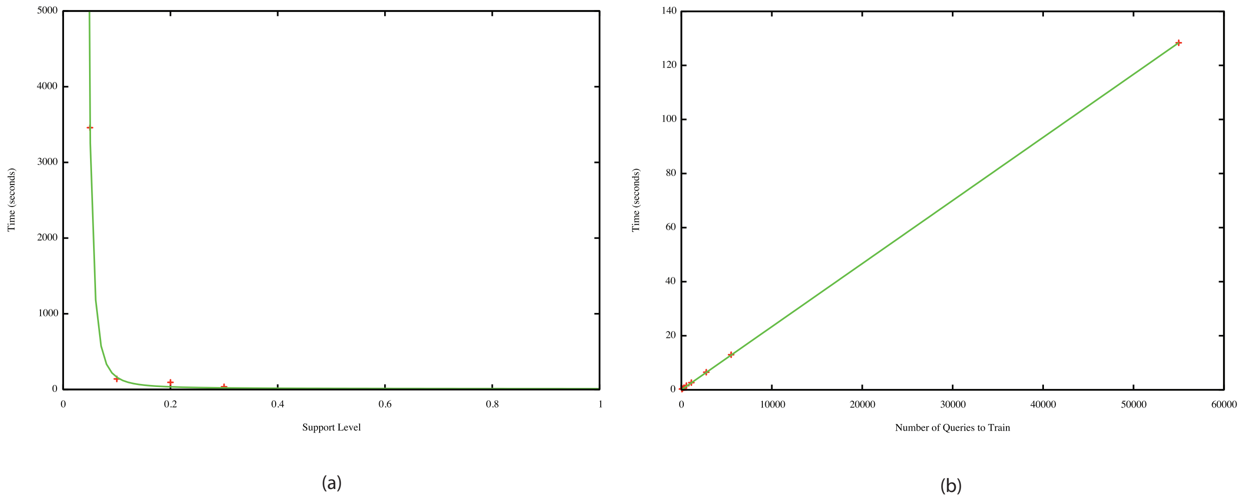


Figure 8.10: **Time Performance of Association Rule Mining**
(a) with respect to the minimum support threshold selected (b) with respect to the size of the training set

The effect of increasing training set sizes was also investigated by running the analysis on larger and larger sets, keeping the support threshold constant. The larger sets in this case were created by duplicating samples in the original set. The results of this experiment is seen in Figure 8.10 part (a), confirming that the analysis scales linearly with the size of the training set (i.e., the repository of previous queries).

While the time performance of the analysis is important, it is not critical to the online operation of the system. The analysis of previous queries will be done periodically and only the rules derived will be used in day to day operation. Therefore, the online processing cost of adding constraints to a joined relation is that of looking up relevant rules from a rulebase. This can be done in many ways such as using a hash index. Regardless of the implementation details, the time cost of processing is essentially only dependent on the number of rules in the rulebase, which in turn will, more than anything, depend on the application domain.

8.4 Candidate Re-encapsulation

The genetic programming approach described in Chapter 7 has been implemented as a prototype using Common Lisp with PostgreSQL as the DBMS. The system was tested on the TPC-H [89] benchmark database. The original TPC-H database is a synthetic database composed of 9 relations and describes the inventory control and sales database of a manufacturing operation. Since the initial assumption is a universal single relation, 7 of these 9 relations were joined over their foreign keys. The resulting join was projected so as to remove the unique attributes and attributes which were semantically redundant. The resulting relation is composed of 25 attributes 12 of which are nominal, 9 numeric and 3 are text attributes. The TPC-H benchmark allows the generation of databases of arbitrary size. Relations of size 10 MB and 100 MB were synthesized in order to study scaling.

The targets to be answered intentionally were initially generated as queries. Each query was generated by selecting random attributes and random values for those attributes using equal probabilities. The query trees were generated using the Probabilistic Tree Creation (PTC2) algorithm [62]. The PTC2 algorithm allows the generation of random queries with precisely defined size. One of the factors examined in the experiments performed was the effect of the complexity of the original query on the accuracy of the intensional answering. Queries ranging from single predicate selections up to combinations of 5 predicates were generated. These queries were in turn converted into targets by executing them on the database and feeding the resulting extensions as target extensions to the system. Table 8.7 gives the number of observations (i.e., runs with different random targets) performed for each factor.

Table 8.7: **Breakdown of Experiments**

		Number of Targets				
		Query Complexity				
DB size	Population Size	1p	2p	3p	4p	5p
10MB	40	200	200	200	200	200
100MB	40	200	200	200	200	200

8.4.1 Accuracy of Answers

Table 8.8: **Overall Error in Ranking**

Mean	0.946
1st Quartile	0.972
2ndQuartile	1.0
Variance	0.01
Minimum	0.45

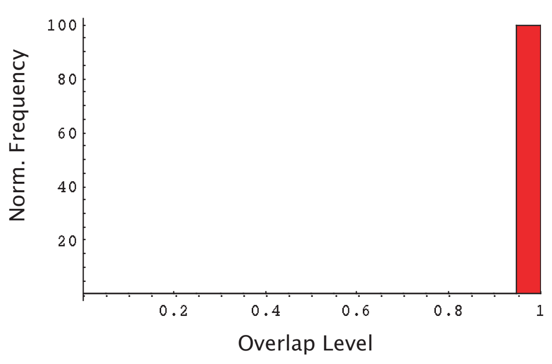
Table 8.8 summarizes the characteristics of fitnesses obtained as a whole. Note that the fitness is defined as the complement e-measure ($E^{'}$) which equals unity for the perfect answer. As can be seen from the summary, for more than half of all targets, the system found the perfect intensional answer, with a mean accuracy of 94.6 %.

The complexity of the original query that generated the target extension seems to have a significant effect on the success of the system. More complex targets result in a noticeable decrease in accuracy of the intensional answers. The mean accuracies for 1 through 5 predicate queries can be seen in Table 8.9 and the distributions of accuracy as a function of complexity are given in Figure 8.11.

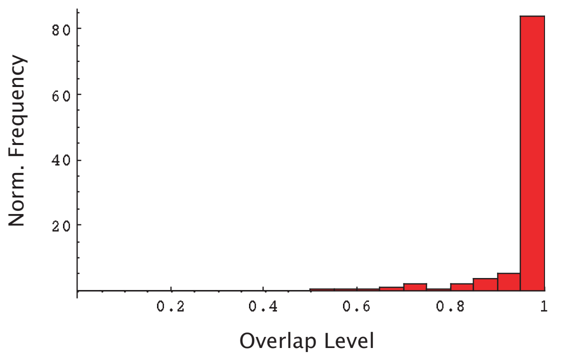
Table 8.9: **Analysis of Accuracy and Concision with respect to Target Complexity**

Target Complexity	Mean $E^{'}$	% Less concise	% Equally Concise	% More Concise
1 Predicate	0.998	1.21	98.79	N/A
2 Predicate	0.969	2.34	84.21	13.45
3 Predicate	0.950	9.03	78.61	12.36
4 Predicate	0.928	17.84	56.48	25.68
5 Predicate	0.877	17.56	47.52	34.92

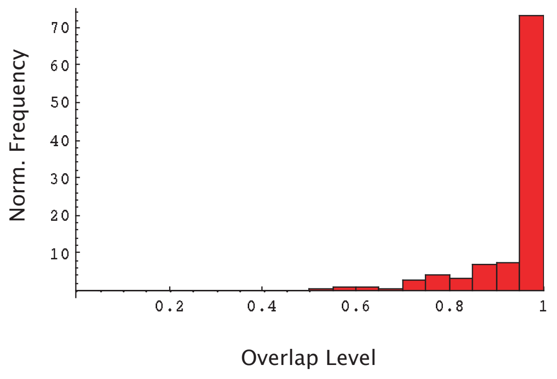
The other interesting result shown in Table 8.9 is that the system manages to generate intensional answers that are most of the time as concise as the original query that generated the target extension. Especially for more complex targets there is an increasing chance that the system will actually improve on the original in terms of the concision. Remember that



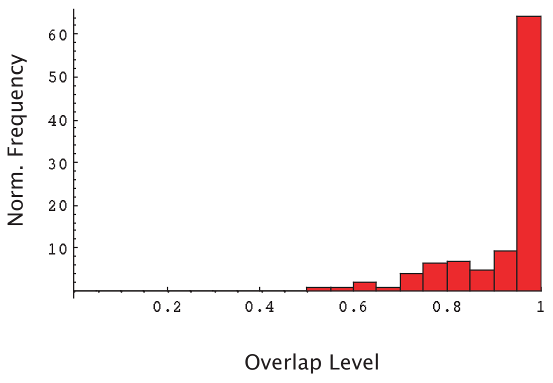
(a) 1 predicate Targets



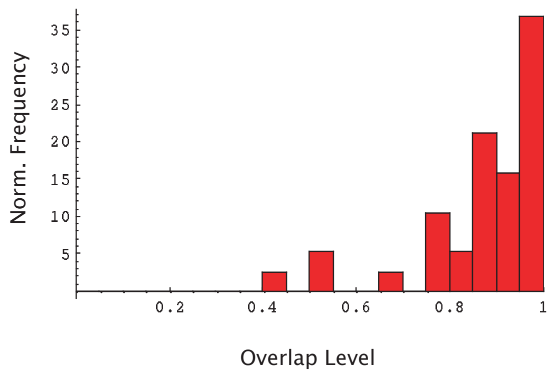
(b) 2 predicate Targets



(b) 3 predicate Targets



(c) 4 predicate Targets



(d) 5 predicate Targets

Figure 8.11: **Answer Accuracies for Targets of Various Complexities**

the system doesn't see the original queries but works only with the extensional forms of those queries.

8.4.2 Time Complexity and Speed

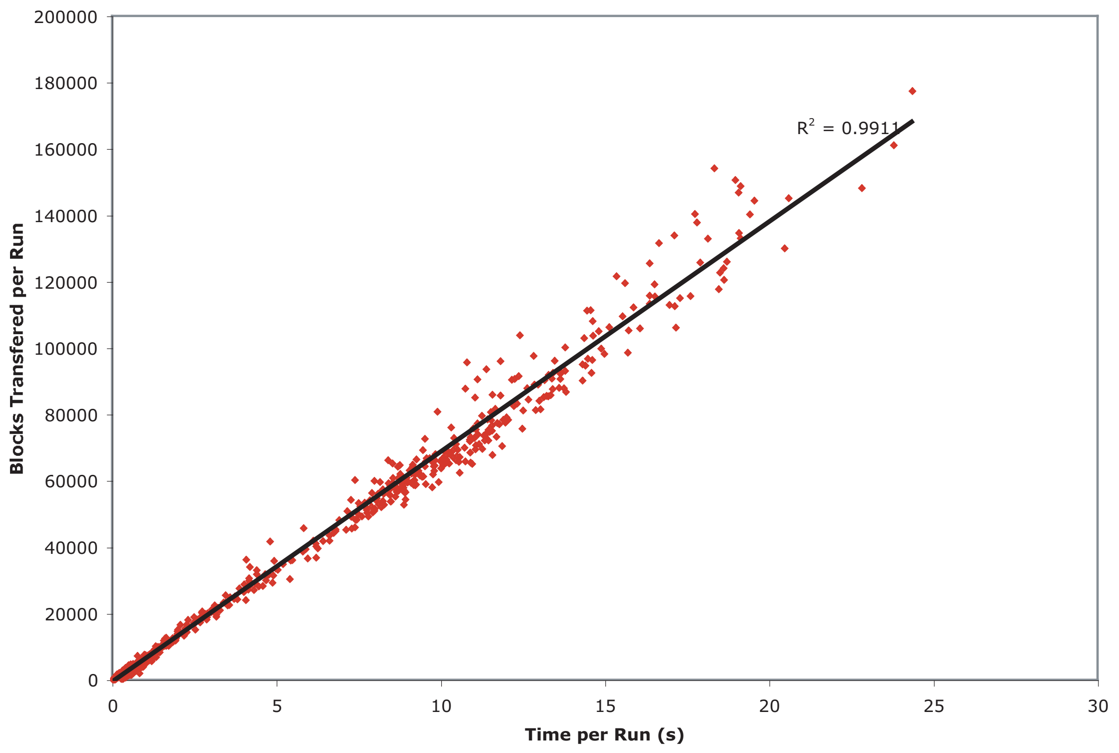


Figure 8.12: **Time vs. DBMS File Activity**

The primary bottleneck for the system is predictably the DBMS. Especially the disk I/O portion of the process seems to dominate the rate at which the system operates. Figure 8.12 plots each of the experiments performed with respect to the time it took to finish the run against the number of blocks retrieved by the DBMS during the run. There is a definite linear proportion between the time it takes to retrieve the data required from the disk and time it takes to finish a run. As added evidence, since each disk block in question is 4 KB, the slope of the line in Figure 8.12 can be found to be 23.7 MB/s. This is incidentally the approximate average disk I/O throughput for the commodity PC[4] used during the experiments.

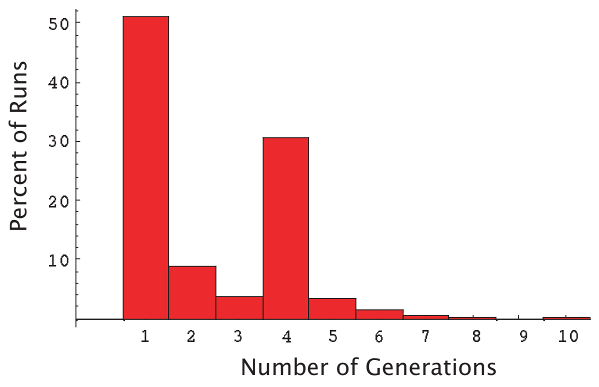


Figure 8.13: **Distribution of Runs in Terms of Generations Required**

Figure 8.13 describes the distribution of the performed runs with respect to the number of generations they require to converge. The mean number of generations required is 2.339. This information, along with the amount of time required per generation can be used to compute the time required for an average run. The average run time for the 10 MB database is 3.76 seconds whereas the average for the 100 MB database is 61.32 seconds. The relationship of the average run times with respect to number of generations is given in Figure 8.14.

Note that the time required as a function of number of generations is actually a weak quadratic. Logically one would expect the relationship to be linear. However, as mentioned in the previous section, the individuals grow larger as generations advance. Therefore, although the number of individuals processed per generation is identical the processing done for later generations is actually longer due to the larger and more complex individuals involved. So, theoretically the square term would start to dominate after 20 generations. Since one never gets that far in reality, the behavior can be regarded as linear in practice.

[4]The machine in question is a 1.8 Ghz AthlonXP CPU with 256MB RAM running FreeBSD on an 80 GB IDE drive with ReiserFS

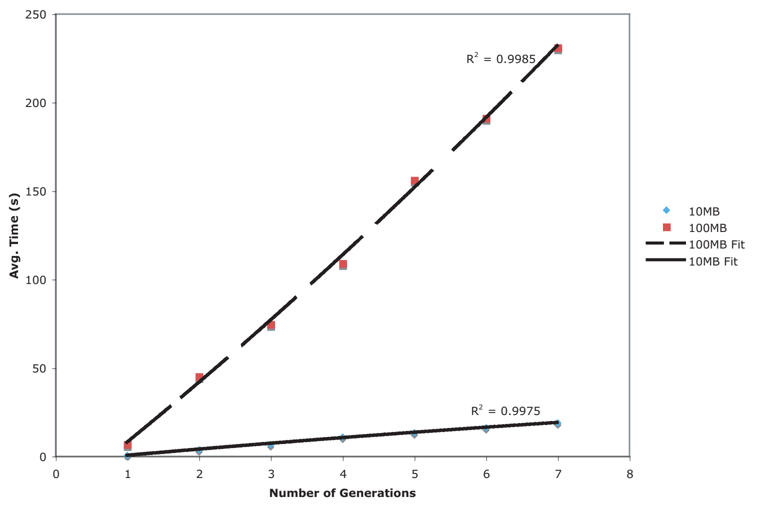


Figure 8.14: **Time requirement vs. Number of Generations**

It should be noted that the primary reason for weakness of the square term is the caching of intermediate results as explained in Section 3.6. This simple optimization removes a significant load off the DBMS especially for larger queries.

Chapter 9: Conclusion

9.1 Summary of the Dissertation

In this dissertation I have introduced and discussed the problem of *query consolidation* which is a reversal of the well known query decomposition problem. The problem is: Given a set of queries which are the result of the decomposition of a global query across multiple independent databases, can we recreate the global query from these pieces?

I have shown that the problem has no unique answer, even with rather limiting assumptions on the query language used and the way the decomposition is done. I have, however, developed an approach which will generate a number of guesses, ranked in order of likelihood, to explain this set.

This is done by first joining these fragments into a view of the user's general information request. By the nature of the problem, it is demonstrated that there are usually multiple joins to each set of queries. I therefore attempt to prune the immense number of possible explanations and further rank them by a scheme which is designed to prioritize more likely explanations.

Additionally, I have optimized the ranking and generation of candidate explanations so that the process can be executed very efficiently, allowing the incoming queries to be extracted and consolidated in real time, as they arrive. Therefore, the approach can necessarily be used in the intermediate steps of the processing of a global query, evolving the scenario as each new piece arrives.

The dissertation also investigates the problem of isolating sets of queries denoting independent user goals from a continuous sequence, either in real time or from a log. The complexities involved can be classified into two categories. The first category of difficulties is defining the boundaries of a goal. The second difficulty is the determination of the individual parts within a goal, regarding their function in forming the goal. An approach based on conditional random fields was introduced that will label individual queries in the sequence according to their functions. The method also labels the initial queries of each distinct set of queries that constitutes a 'task' of the user. The labels can be decoded very efficiently as each query arrives, presenting a new label sequence with each update. Along with the fast join inference method mentioned above, this allows one to evaluate the most likely information goals of a querying agent as they ask each new question, reevaluating the scenario and coming up with likely solutions as each new query arrives.

I also provide three methods to further enhance the generated consolidations. The first of these is a rule base of associations from previous experience that refines the candidate consolidations with likely constraints, given the information they retrieve (i.e., attributes projected). The second is a chase algoritm that will expand the projections of the possible consolidations with implied information, based on the functional dependency information available in the global schema. Finally, a query re-encapsulation method is introduced that will return to the extension of a consolidation and attempt to 'evolve' a more concise expression that will define the same extension. The re-encapsulation, along with the other two enhancements will allow the person monitoring the query set of a user a different and possibly simpler insight on the actual intent of the user.

All of these methods discussed above have been validated with detailed experimentation on real and synthetic data. The result show that the methods proposed are viable in terms of accuracy. The optimized methods (i.e., join inference and enhancement with global selections) are all shown to be efficient enough to work in real-time. The re-encapsulation, due to the computationally intensive nature of genetic programming, has very good accuracy but is not especially fast. However, the re-encapsulation method is very amenable to parallel computation and with the advent of multiple core processors and clustering, could soon be viable for real time operation as well.

9.2 Contributions

In this dissertation I make the following contributions:

1. The introduction and initial treatment of the problem of query consolidation.
2. An analysis on the bounds of the number of extensions that can be built using a known number of views with known arity, given the binary operations possible.
3. A join inference method that enumerates and ranks, by likelihood, the possible join plans given a set of views.
4. An ad-hoc key inference method that is computationally economical and independent of schema information (i.e., dependent only on the data)
5. A method for segmenting and tagging continuous sequences (i.e., logs) of queries into cliques of related queries using conditional random fields.
6. The application of association analysis to discover relationships between projection sets and selection constraints in queries.
7. A method of encapsulating database subsets with intensional expressions based on genetic programming and Bayesian inference.

9.3 Future Work

There are several avenues of research that would benefit the understanding of the problem and improve the methods described in this dissertation. The schema agnostic join inference method in Sec. 4.2, in its current state, is optimized with assumption that keys are composed of single attributes. While it is generalizable to multiple attribute keys, it does not scale well in such cases. Being that the main advantage of the method is its speed and scalability, a redesign or improvement of the basic algorithm is needed for the more general case.

The sequence analysis (Sec. 5) does not handle interleaved query sequences. The queries are segmented with assumption that two individual goals arrive separated chronologically and do not share components. This assumption is made in order to be able to handle the sequence segmentation problem in a simple and unambiguous manner. While not unjustified, it may indeed need to be relaxed. The possibility of a sequence analysis that can separate interleaved query sequences is a worthwhile research topic. Initially, this could be attempted by a mixture model whereby the log-likelihood of a label sequence may be generated by a mixture of more than one CRF, or similar probabilistic model.

As mentioned in the previous section, the query encapsulation method is not as efficient as the alternatives in literature. The main contender being decision tree classifiers, a genetic programming approach was taken as it better captures small disjuncts in the data [24]. In the case of the present problem, this corresponds the ability to capture the intensional explanation of recursive query plans or the collection of data from many sources. A different encapsulation paradigm may be researched possibly using a boosting approach with several fast but not necessarily very accurate classifiers.

Finally, during this research I have mostly demurred from using explicit semantics. In other words, most of the proposed research does not depend on a wealth of metadata or markup. Almost all of the metadata required is within the underlying global schema $\mathcal{G}$ and consists of information limited to that of a DBMS catalog (e.g. attribute names, types and referential constraints). Furthermore, the approach was designed to estimate the semantics, when not available, from its latent effects on the data itself and thus to require as little human intervention as possible. Over the half decade the present research has been going on however, a good deal of development has been done on semantic enrichment of data sources. Furthermore, metadata is being increasingly added to repositories in real life. Therefore, the independence of the current system from metadata may be relaxed. The use of ontologies and the associated logical inference mechanisms in a query consolidating system should probably be researched, as these will undoubtedly help in eliminating some of the solution multiplicity, enhance the estimation of likelihood of a given consolidation and possibly aid the discovery of simpler intensional explanations.

Bibliography

Bibliography

[1] Abiteboul, S., Hull, R., and Vianu, V. *Foundations of Databases.* Addison-Wesley, 1995, ch. 4, pp. 55–56.

[2] Acar, A. C., and Motro, A. Why is this User Asking so Many Questions?: Explaining Sequences of Queries. In *Proceedings of the 18th Annual IFIP WG 11.3 Working Conference on Data and Applications Security, Sitges, Spain* (2004), Kluwer, pp. 159–176.

[3] Acar, A. C., and Motro, A. Intensional Encapsulations of Database Subsets via Genetic Programming. In *Proceedings of DEXA '05* (2005), pp. 365–374.

[4] Adali, S., Candan, K., Papakonstantinou, Y., and Subrahmanian, V. Query Caching and Optimization in Distributed Mediator Systems. In *Proceedings of ACM SIGMOD '96, Montreal, Canada* (1996), ACM Press, pp. 137–148.

[5] Adali, S., and Emery, R. A Uniform Framework for Integrating Knowledge in Heterogenous Knowledge Systems. In *Proceedings of IEEE ICDE '95, Taipei, Taiwan* (1995), pp. 513–520.

[6] Adam, N. A New Dynamic Voting Algorithm for Distributed Database Systems. *IEEE Transactions on Knowledge and Data Engineering 6*, 3 (June 1994), 470–478.

[7] Agarwal, R. C., Aggarwal, C. C., and Prasad, V. V. V. A Tree Projection Algorithm for Generation of Frequent Item Sets. *J. Para. Distr. Comput. 61*, 3 (2001), 350–371.

[8] Agrawal, R., Imielinski, T., and Swami, A. N. Mining Association Rules between Sets of Items in Large Databases. In *Proceedings of ACM SIGMOD '93, Washington D.C.* (1993), P. Buneman and S. Jajodia, Eds., ACM Press, pp. 207–216.

[9] Agrawal, R., Manilla, H., Srikant, R., Toivonen, H., and Verkamo, A. I. Fast Discovery of Association Rules. In *Advances in Knowledge Discovery and Data Mining*, U. M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, Eds. AAAI/MIT Press, 1996, ch. 12, pp. 307–328.

[10] Agrawal, R., and Srikant, R. Fast Algorithms for Mining Association Rules. In *Proceedings of VLDB '94* (1994), J. B. Bocca, M. Jahrke, and C. Zaniolo, Eds., pp. 487–499.

[11] Agrawal, S., Chaudhuri, S., and Das, G. DBXplorer: Enabling keyword search over relational databases. In *Proceedings of ICDE 02* (2002), IEEE Computer Society, pp. 5–16.

[12] Angluin, D. A Note on the Number of Queries Needed to Identify Regular Languages. *Information and Control 51*, 1 (1981), 76–87.

[13] Baeza-Yates, R. A., and Ribeiro-Neto, B. *Modern Information Retrieval.* ACM Press / Addison-Wesley, 1999, p. 82.

[14] Beeferman, D., and Berger, A. Agglomerative Clustering of a Search Engine Query Log. In *Proceedings of Knowledge Discovery and Data Mining* (2000), pp. 407–416.

[15] Beeri, C., Fagin, R., Maier, D., and Yannakakis, M. On the Desirability of Acyclic Database Schemes. *Journal of the ACM 30*, 3 (1983), 479–512.

[16] Berlin, J., and Motro, A. Autoplex: Automated Discovery of Contents for Virtual Databases. In *In Proc. COOPIS-01* (2001), pp. 108–122.

[17] Berlin, J., and Motro, A. Database Schema Matching Using Machine Learning with Feature Selection. In *Proceedings of CAiSE 02* (2002), pp. 452–466.

[18] Bilmes, J. A Gentle Tutorial on the EM Algorithm and its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models. Tech. Rep. ICSI-TR-97-021, University of Berkeley, 1997.

[19] Buneman, P. Semistructured Data. In *Proceedings of ACM PODS 97* (1997), pp. 117–121.

[20] Burdick, D., Calimlim, M., and Gehrke, J. MAFIA: A Maximal Frequent Itemset Algorithm for Transactional Databases. In *Proceedings of IEEE ICDE '01, Heidelberg, Germany* (2001), IEEE Computer Society, pp. 443–452.

[21] Calvanese, D., Giacomo, G. D., and Lenzerini, M. Description Logics for Information Integration. In *Computational Logic: From Logic Programming into the Future*, Lecture Notes in Computer Science. Springer-Verlag, 2001.

[22] Cardiff, J., Catarci, T., and Santucci, G. Semantic Query Processing in a Heterogeneous Database Environment. *Journal of Intelligent and Cooperative Information Systems 6*, 2 (1997), 151–192.

[23] Carvalho, D. R., and Freitas, A. A. A Genetic Algorithm-based Solution for the Problem of Small Disjuncts. In *Lecture Notes in Artificial Intelligence No.1910* (2000), pp. 345–352.

[24] Carvalho, D. R., and Freitas, A. A. A Hybrid Decision Tree/Genetic Algorithm for Coping with the Problem of Small Disjuncts in Data Mining. In *Proc. 2000 Genetic and Evolutionary Computation Conf* (2000), pp. 1061–1068.

[25] Castano, S., and De Antonellis, V. A Schema Analysis and Reconciliation Tool Environment. In *Proceedings of IDEAS 99* (1999).

[26] Catarci, T. What Happened When Database Researchers Met Usability. *Information Systems 25*, 3 (May 2000), 177–212.

[27] Chen, M.-S., Park, J. S., and Yu, P. S. Efficient Data Mining for Path Traversal Patterns. *IEEE Transactions on Knowledge and Data Engineering 10*, 2 (1998), 209–221.

[28] Cooley, R., Mobasher, B., and Srivastava, J. Data Preparation for Mining World Wide Web Browsing Patterns. *Knowledge and Information Systems 1*, 1 (1999), 5 – 32.

[29] Cramer, N. L. A Representation for the Adapative Generation of Simple Sequential Programs. In *Proceedings of the 1st International Conference on Genetic Algorithms* (1985), pp. 183–187.

[30] DeGiacomo, G. Intensional Query Answering by Partial Evaluation. *J. Intell. Info. Sys. 7*, 3 (1996), 205–233.

[31] Doan, A., Domingos, P., and Halevy, A. Reconciling Schemas of Disparate Data Sources: A Machine Learning Approach. pp. 509–520.

[32] Duschka, O. M. *Query Planning and Optimization in Information Integration.* PhD thesis, Stanford University, 1997.

[33] Duschka, O. M., and Genesereth, M. R. Answering Recursive Queries using Views. In *Proceedings of ACM PODS 97* (1997), pp. 109–116.

[34] Flockhart, I. W., and Radcliffe, N. J. GA-MINER: Parallel Data Mining with Hierarchical Genetic Algorithms - Final Report. Tech. rep., The University of Edinburgh, 1995.

[35] Freitas, A. A. A Survey of Evolutionary Algorithms for Data Mining and Knowledge Discovery. In *Advances in Evolutionary Computing: Theory and Applications*, A. Ghosh and S. Tsutsui, Eds. Springer-Verlag, 2003, pp. 819–845.

[36] Garcia-Molina, H., Papakonstantinou, Y., Quass, D., Rajaraman, A., Sagiv, Y., Ullman, J. D., Vassalos, V., and Widom, J. The TSIMMIS Approach to Mediation: Data Models and Languages. *J. Intell. Info. Sys. 8*, 2 (1997), 117–132.

[37] Godfrey, P., and Gryz, J. Semantic Query Caching for Heterogeneous Databases. In *Knowledge Representation Meets Databases* (1997), pp. 6.1–6.6.

[38] Grant, J., and Jacobs, B. E. On the Family of Generalized Dependency Constraints. *Journal of the ACM 29*, 4 (1982), 986–997.

[39] Han, J., Pei, J., and Yin, Y. Mining Frequent Patterns without Candidate Generation. In *Proceedings of ACM SIGMOD '00, Dallas, TX* (2000), W. Chen, J. Naughton, and P. A. Bernstein, Eds., ACM Press, pp. 1–12.

[40] He, D., and Goker, A. Detecting Session Boundaries from Web User Logs. In *Proceedings of the BCS-IRSG 22nd Annual Colloquium on Information Retrieval* (2000).

[41] Hipp, J., Guntzer, U., and Nakhaeizadeh, G. Algorithms for Association Rule Mining – A General Survey and Comparison. *SIGKDD Explorations 2*, 1 (July 2000), 58–64.

[42] Hristidis, V., and Papakonstantinou, Y. DISCOVER: Keyword Search in Relational Databases. In *Proceedings of VLDB '02, San Fransisco, CA* (2002), Morgan Kaufmann, pp. 670–681.

[43] Imielinski, T. Intelligent Query Answering in Rule Based Systems. *Journal of Logic Programming 4*, 3 (1987), 229–257.

[44] Jin, R., Yan, R., Zhang, J., and Hauptmann, A. A Faster Iterative Scaling Algorithm for Conditional Exponential Model. In *Proceedings of the 20th Int. Conf. on Machine Learning* (2003), pp. 282–289.

[45] Joachims, T. Unbiased Evaluation of Retrieval Quality using Clickthrough Data. Tech. rep., Cornell University, Department of Computer Science, 2002.

[46] Kapoor, S., and Ramesh, H. Algorithms for Enumerating All Spanning Trees of an Undirected Graph. *SIAM Journal on Computing 24*, 2 (1995), 247–265.

[47] Kindermann, R., and Snell, J. *Markov Random Fields and their Applications.* American Mathematical Society Providence, RI, 1980.

[48] Kirk, T., Levy, A. Y., Sagiv, Y., and Srivastava, D. The Information Manifold. In *Proceedings of the AAAI Spring Symp. on Information Gathering from Heterogeneous, Distributed Environments* (1995), pp. 85–91.

[49] Kivinen, J., and Mannila, H. Approximate Inference of Functional Dependencies from Relations. *Theoretical Computer Science 149*, 1 (1995), 129–149.

[50] Koza, J. R. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, 1992.

[51] Koza, J. R. *Genetic Programming II: Automatic Discovery of Reusable Programs.* MIT Press, 1994.

[52] Kullback, S., and Leibler, R. A. On Information and Sufficiency. *Annals of Mathematical Statistics 22*, 1 (1951), 79–86.

[53] Lafferty, J., McCallum, A., and Pereira, F. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *Proceedings of the 18th Int. Conf. on Machine Learning* (2001), pp. 282–289.

[54] Larson, J., Navathe, S., Elmasri, R., and Honeywell, M. A Theory of Attributed Equivalence in Databases with Application to Schema Integration. *IEEE Transactions on Software Engineering 15*, 4 (1989), 449–463.

[55] Lenzerini, M. Data Integration: A Theoretical Perspective. In *Proceedings of ACM PODS '02, Madison, WI* (2002), ACM Press, pp. 233–246.

[56] Levy, A. Y., Rajaraman, A., and Ordille, J. J. Querying Heterogeneous Information Sources using Source Descriptions. *Proceedings of VLDB 96* (1996), 251–262.

[57] Levy, A. Y., Rajaraman, A., and Ullman, J. D. Answering Queries Using Limited External Query Processors (Extended Abstract). In *Proceedings of the 15th ACM Symposium on Principles of Database Systems* (1996), pp. 227–237.

[58] Levy, A. Y., and Sagiv, Y. Semantic Query Optimization in Datalog Programs. In *Proceedings of PODS* (1992), pp. 163–173.

[59] Li, W., and Clifton, C. SEMINT: A Tool for Identifying Attribute Correspondences in Heterogeneous Databases using Neural Networks. *Data & Knowledge Engineering 33*, 1 (2000), 49–84.

[60] Liu, D., and Nocedal, J. On the Limited Memory BFGS Method for Large Scale Optimization. *Mathematical Programming 45*, 1 (1989), 503–528.

[61] Luke, S. *Issues in Scaling Genetic Programming: Breeding Strategies, Tree Generation, and Code Bloat.* PhD thesis, Department of Computer Science, University of Maryland, College Park, Maryland, 2000.

[62] Luke, S. Two Fast Tree-Creation Algorithms for Genetic Programming. *IEEE Trans. Evol. Comp. 4*, 3 (2000), 274–283.

[63] Maier, D., and Ullman, J. D. Maximal Objects and the Semantics of Universal Relation Databases. *ACM Transactions on Database Systems 8*, 1 (1983), 1–14.

[64] Maier, D., Ullman, J. D., and Vardi, M. Y. On the Foundations of the Universal Relation Model. *ACM Trans. Database Syst. 9*, 2 (1984), 283–308.

[65] Manolescu, I., Florescu, D., and Kossmann, D. Answering XML queries on hetergenous data sources. In *Proceedings of the 27th Int. Conference on Very Large Data Bases, Rome, Italy* (2001), pp. 241–250.

[66] Mason, T., and Lawrence, R. INFER: A relational query language without the complexity of sql. In *Proceedings of CIKM 05* (2005), pp. 241–242.

[67] McCallum, A. Efficiently Inducing Features of Conditional Random Fields. In *Proceedings of the 19th Annual Conference on Uncertainty in Artificial Intelligence (UAI-03)* (2003), pp. 403–411.

[68] Miller, G. A. The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information. *Psychological Review 63*, 2 (1956), 81–97.

[69] Motro, A. Superviews: Virtual Integration of Multiple Databases. *IEEE Transactions on Software Engineering SE-13*, 7 (July 1987), 785 – 798.

[70] Motro, A. Using Integrity Constraints to Provide Intensional Responses to Relational Queries. In *Proceedings of VLDB 89, the 15th International Conference on Very Large Data Bases* (1989), pp. 237–246.

[71] Motro, A. Intensional Answers to Database Queries. *IEEE Transactions on Knowledge and Data Engineering 6*, 3 (June 1994), 444–454.

[72] Motro, A. Multiplex: A Formal Model for Multidatabases and Its Implementation. In *Proceedings of NGITS 99, Fourth International Workshop on Next Generation Information Technologies and Systems, Zichron Yaacov, Israel* (1999), Lecture Notes in Computer Science No. 1649, Springer-Verlag, pp. 138–158.

[73] Motro, A., and Anokhin, P. Fusionplex: resolution of data inconsistencies in the integration of heterogeneous information sources. *Information Fusion 7*, 2 (2006), 176–196.

[74] Motro, A., Anokhin, P., and Acar, A. C. Utility-based Resolution of Data Inconsistencies. In *Proceedings of International Workshop on Information Quality in Information Systems, ACM SIGMOD '04, Paris, France* (2004), ACM Press, pp. 35–43.

[75] Papakonstantinou, Y., Abiteboul, S., and Garcia-Molina, H. Object Fusion in Mediator Systems. In *Proceedings of the 22nd Int. Conf. on Very Large Data Bases* (1996), pp. 413–424.

[76] Parekh, R., and Honavar, V. An Incremental Interactive Algorithm for Regular Grammar Inference. In *Proceedings of ICGI 96* (1996).

[77] Pazzani, M. J. An Iterative-Improvement Approach for the Discretization of Numeric Attributes in Bayesian Classifiers. In *Proceedings of AAAI KDD 95* (1995), pp. 228–233.

[78] Pinto, D., McCallum, A., Wei, X., and Croft, W. Table Extraction using Conditional Random Fields. *Proceedings of ACM SIGIR 03* (2003), 235–242.

[79] Pirotte, A., Roelants, D., and Zimanyi, E. Controlled Generation of Intensional Answers. *IEEE Trans. Knowl. Data. Eng. 3*, 2 (1991), 221–236.

[80] Pottinger, R., and Halevy, A. MiniCon: A Scalable Algorithm for Answering Queries using Views. *The VLDB Journal 10*, 2 (2001), 182–198.

[81] Qian, X. Query Folding. In *Proceedings of IEEE ICDE 96* (1996), pp. 48–55.

[82] Rabiner, L. R. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE 77*, 2 (1989), 257–286.

[83] Rahm, E., and Bernstein, P. A. A Survey of Approaches to Automatic Schema Matching. *The VLDB Journal 10*, 4 (2001), 334–350.

[84] Savasere, A., Omiecinski, E., and Navathe, S. B. An Efficient Algorithm for Mining Association Rules in Large Databases. In *Proceedings of VLDB '95, Zurich, Switzerland* (1995), U. Dayal, P. M. D. Gray, and S. Nishio, Eds., Morgan Kaufmann, pp. 432–444.

[85] Shum, C. D., and Muntz, R. R. Implicit Representation for Extensional Answers. In *Proc. Second Int. Conf. on Expert Database Systems* (1988), pp. 497–522.

[86] Silberschatz, A., Korth, H. F., and Sudershan, S. *Database System Concepts.* McGraw-Hill, Inc., New York, NY, USA, 1998.

[87] Sloan Digital Sky Survey. SkyServer. http://www.sdss.org., 2007.

[88] Tan, P.-N., Kumar, V., and Srivastava, J. Selecting the Right Interestingness Measures for Association Patterns. In *Proceedings of ACM SIGKDD '02, Edmonton, Canada* (2002), ACM Press, pp. 32 – 41.

[89] Transaction Processing Performance Council. TPC Benchmark H Rev 2.6.1. Standard Specification, 2007.

[90] Univ. of Goettingen Ins. for Informatics. The MONDIAL Database. http://www.dbis.informatik.uni-goettingen.de/Mondial/, 2007.

[91] Wald, J. A., and Sorenson, P. G. Resolving the Query Inference Problem using Steiner Trees. *ACM Trans. Database Syst. 9*, 3 (1984), 348–368.

[92] Wallach, H. Efficient Training of Conditional Random Fields. Master's thesis, University of Edinburgh, 2002.

[93] Yao, Q., Huang, X., and An, A. A Machine Learning Approach to Identifying Database Sessions Using Unlabeled Data. In *Proceedings of Conf. on Data Warehousing and Knowledge Discovery* (2005), pp. 254–264.

Curriculum Vitae

Aybar C. Acar was born in 1978 in Ankara, Turkey. He received his B.S. and M.S. degrees in Chemical Engineering from the Middle East Technical University in 1999 and 2001, respectively. He has worked as a research assistant in the Chemical Engineering Department at METU, and a teaching assistant at the Computer Science Department at the Volgenau School of Information Technology and Engineering. He has also worked as a researcher at the METU Software Research and Development Center and a software designer and consultant at ACP Inc. in Melville, NY.