Engineering truly automated data integration and translation systems.

by

Robert H. Warren

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2007

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Robert H. Warren

Abstract

This thesis presents an automated, data-driven integration process for relational databases. Whereas previous integration methods assumed a large amount of user involvement as well as the availability of database meta-data, we make no use of meta-data and little end user input. This is done using a novel join and translation finding algorithm that searches for the proper key / foreign key relationships while inferring the instance transformations from one database to another. Because we rely only on the relations that bind the attributes together, we make no use of the database schema information. A novel searching method allows us to search the database for relevant objects without requiring server side indexes or cooperative databases.

Acknowledgements

This thesis represents a 'short' synopsis of several years worth of work during my doctorate degree. The work would have been impossible without the support of my advisor Dr. Frank Wm. Tompa, my committee members Dr. Gord Cormack and Dr. Grant Weddell and the administrative staff of the School of Computer Science. My thanks also to Dr. Stefan Steiner and Dr. Divesh Srivastava for their time and patience as my external examiners. All of them must have at one time or another wondered exactly what they had gotten into.

I am especially grateful to my family for supporting my project during a difficult time and putting up with me.

A special mention must go out to Mike Patterson and the other staff of the CS Computing Facility who went to heroic lengths to support my work, and didn't blink when asked for a terabyte of storage on short notice. Dana Wilkinson and Benjamin Korvemaker also helped me greatly with computing problems and checking my wild ideas. David Evans gets a special mention for listening to my many rants while sitting in a ski hill chairlift.

This work was partially financially supported by the Ontario Graduate Scholarship fund, NSERC and also made possible by the facilities of the Shared Hierarchical Academic Research Computing Network (SHARCNET) with the help of Rob Schmidt.

Invariably, thesis acknowledgements end with a vague statement that the people that made the thesis possible are too numerous to list. I'll break with that tradition and list them below with many thanks.

Robert Warren, Waterloo, 2007

Banks, Boucher, Buettcher, Buckley, Butler, Champaign, Campbell, Charlin, Clarke, Clift, Collins-Thompson, Collison, Cormack, DeBeaudrap, Deschamps, Braehler, Du Toit, Echtner, Evans, Fishbein, Fontaine, Folland, Gurak, Haarslev, Harji, Harman, Hindle, Hinek, Hines, Honeyford, Huang, Hudek, H3, Johnson, Jordan, Juarez-Dominguez, Kapser, Kerr, Krischer, Kroon, Kwan, Kwok, Korvemaker, Koutouki, Krzysztof, Lamontagne, Lavad, Larke, Leroux, Lester, Leurer, Lynam, Mackinnon, McBay, McPhearson, Miranda, Mirkina, Mills, Morrison, Mustin, Niehage, Oldford, Patterson, Perez-Delgado, Peterson, Prime, Poznanski, Quimper, Reardon, Research Net, Rush, Russell, Schmidt,

Smith, Spanos, Spriggs, Spitzer, Strommer, Suchostawski, Talbot, Tausky, Terra, Tompa, VanVelzen, Warren, Warnecke, Weddell, Wilkinson, Wong, Wonta, Zaverucha, Ziembicka.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Database integration has been defined as "the problem of providing unified and transparent access to a collection of data stored in multiple, autonomous, and heterogeneous data sources." [69]

The dual pressures of larger databases containing more data, along with the increasing availability of low-cost bandwidth, is creating the conditions under which it is increasingly worthwhile to integrate disparate information systems.

Most of the previous work in the area concerned itself with the theories required to reconcile classical database theory to that of exchanging data between databases. Most recently, work done by the community has extended itself to supporting the automated integration of the databases.

This thesis concerns itself with extending some of the methods and advancing the field in terms of its ability to deal with very large databases of unknown and obscure design.

1.1 Motivation and caveats

Most work still relies on the assumption that a human being will be involved in the integration process and will fetch and reconcile any difficulty in managing the transfer of data or query. Automating this task is now a necessity because of the cost of manpower and the rate at which the integration is expected to occur.

Another issue is that with pervasive networks of telecommunications, large numbers of

application specific databases and the advent of the problem of the "deep" or "hidden" web [94], the process of integration becomes an opportunistic one: the information needed for a particular objective exists, but the location and accessibility of any particular database is transient in nature.

In this model it is impractical to assume that an expert will be available for each integration problem. Algorithmic methods might be better overall solutions, as they enable us to attempt to take advantage of an opportunity to acquire the data of interest.

In many cases, much of the database experts' time is spent on clerical processes of little value-add, such as converting data from one representation to another or searching for the correct information within the database. It is this clerical process that we aim to automate in our research.

1.1.1 A data-driven approach to database integration.

Several projects have begun to tackle the integration problem from a top-down perspective, using the schema and database information to design a solution tailored to the source databases. Instead, we use a bottom-up approach that is data-driven in that it uses the information contained in the database instead of the schema to make integration decisions. In effect, we search the databases themselves for matches and infer translations without depending on full schema information.

Seligman et al. [102] have published a survey that ranked the acquisition of knowledge about the data sources as the data integration step requiring the most effort. Large and complex industrial database schemas with over 10,000 tables and over 1,600 attributes per table are not unheard of, and even with good documentation, the search for the correct information is time consuming. Ventrone and Heiler [109] similarly reported that GTE required on average 4 hours of work to fully understand a data element when the designers were not available.

The data driven approach can be criticised for its non-trivial computational cost, and in many situations this would be a serious concern. However, we pursue this line of research for applications where computational resources are easier to acquire than additional domain experts.

Problems that drive the demand for data-driven data integration include:

Incomplete database schema documentation and design instability: These are key problems in integration in that the schema information required to perform the integration must be available. Database designs may have been lost or be stored in a format that is either impossible to recover or in a format that is not machine readable. If this information is not available, then we must rely on the data within the database itself. Furthermore, in the case of very active and dynamic databases, the schema is likely to change very often or to be abused as the data requirements overtake the specifications of the schema. Similarly, legacy databases can also develop these situations as the design knowledge is lost while the data-centric knowledge remains, due to day-to-day use.

Unavailability of integration support: Integration is a complex process that requires significant information about the design of the databases in use. There are situations where the database design knowledge is separated from the access to the data, with two different domain experts required.

In these cases, we must be able to integrate the database based on its actual behaviour instead of its (inaccessible) documented behaviour. The canonical example of this is a web database; accessible through a web page or web services, but never intended to be integrated into a third party service. The database design information is not only unavailable from the designers, but in some cases the data source schema is effectively obfuscated because of the specificity of the intended application. While we do not assume that these databases were designed by an adversarial actor, it can be said that this is effectively non-cooperative integration.

Increased data volume and complexity: A side-effect of the increasing complexity and volume of information being stored within databases is that trivial assumptions about databases need to be revisited. Many current schema based database integration engines make the assumption that the schema labelling is semantically significant.

As the schema size of the databases increase, so does the complexity of the labels given to schema objects. Hence, it is difficult to account for the difference between different schema objects named "shipping_address, shipping_address_dock and ship-

ping_address_current" and to match them to another database schema.

Similarly, when exceeding a language threshold the namespace becomes exhausted and labels are only used as keys for a data definition library. In this situation, the schema labels become useless without additional documentation (e.g.: table `04GHS46F5`) and cannot be matched against another database with a different definition library.

While the size of the namespace grows exponentially, meaningful human-readable labelling can only grow at a much slower pace that limits our ability to describe the information accurately.

On the other hand, two developments support the prospect of data-driven integration:

Standardised interfaces to database systems: Much work has been done to develop standardised interfaces to facilitate the transfer of the data. Application programming interfaces, such as JDBC and ODBC, make it now possible to retrieve information easily from any table or column within most databases.

In practice, the use of a standardised interface to access diverse database management systems implies that a mapping is used between the standard interface and each database system.

Perversely, uneven implementations mean that in practice, it is often easier to retrieve the actual data from the database instead of the schema information (relationships, constraints, etc.) Since most current integration approaches require this information, it is very likely that we could be faced with the frustrating situation where the database data could be available, but the integration method would fail.

Therefore, we require an additional theory that will allow us to integrate a database based on its observed data, as well as the reported schema.

Increased opportunity in data acquisition: Database integration research is implicitly being approached as a one-shot process by the community. It is essentially seen as a capital project with a resource cost that is assumed to be an investment that will yield returns based on the data accessed over time.

Because the resources required to perform the integration are relatively high, only the data sources that will yield a long term return will be considered.

An automated, data-driven, integration solution allows us to exploit short lived data sources and / or data sources who's contents could never justify the logistical effort required by a traditional, human-driven solution.

1.1.2 Further considerations

As an added complication to the considerations already mentioned, we also face the problem of the locality of the databases versus the integration process. In an ideal case, we would operate the database locally as a simple repository without problems.

Yet, it does stand to reason that the databases being integrated were not initially integrated because they were all separately stored and independently managed. This implies that there exists a logical, if not physical, separation between the databases and the integration process must use a specific interface with certain limitations, such as bandwidth.

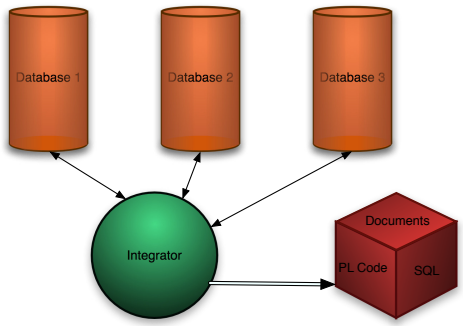

Figure 1.1: The separation between the integration process and the targeted databases means that query results have a cost.

This places a restriction on the types of queries that we can and should pose to the databases, since we have limited resources. We do not make use of a cost model within this work because we are working at a coarser granularity. We simply state as an axiom that in all cases the databases are too large in their size, or otherwise too costly, to enable us to transfer all of the data locally to the integrator. Instead, the query facilities must be used to their maximum potential.

We make an explicit assumption that all databases are accessible using a basic subset of SQL99 [58]. While other query facilities are in use, it is the one of the most widespread, and its limited number of query constructs provides a base case for the integration methods.

1.2 Basic integration scenario

Let us assume that we have a local database D^{local} of movies and TV programs that we own, with information about known movies and TV series. Furthermore, we assume for convenience that this database is well-formed and can be projected to a single relation R^{local}, as represented in Table 1.1.

Attribute Name	Description	Example_1	Example_2
Title	The name of the media	The Terminator	Star Trek
SubTitle	The sub-title/name of the media	-	Spock's Brain
Year	The year that the title was released	1984	1968
Type	The type of movie (series, Movie)	Movie	Series
Genre	The story genre of the title	Action	Sci-fi
Description	The plot summary of the feature	...	...
Cast	The name of an actor in the feature	Arnold Schwarzenegger	William Shatner

Table 1.1: Examples of the types of data contained within the MythTV and IMDB databases.

Let us assume that there exists another database D^{foreign} that also contains information on movies and TV series, among other information. We wish to discover what parts of the database D^{foreign} to import into D^{local} and the specific querying mechanisms required to move and transform the data.

We assume that some pattern of joins of the relations in D^{foreign}, possibly also involving some text transformations of attribute values, will form a single relation that is semantically compatible with the single relation in D^{local}. We also assume that such a transformed relation from D^{foreign} will share some data values with D^{local}.

1.2.1 Database model assumption

There is the question of what assumptions we make about the database model or the specific features of the data in the foreign database. We have already stated our assumption that the size of the database is too large to simply copy all of it over to a local integration client for processing.

In practice, this means that the database schema consists of several thousand relations whose labels do not convey either semantic or structural meaning. The number of relations within the schema requires us to make a choice about what relations are to be integrated, as there are too many to retrieve to the client for local processing.

The number of tuples within each relation is assumed to be large enough that retrieving all tuples to the client is impractical. Of course, in reality not all relations are the same size, and there may even be several relations that could be completely retrieved to the client. However, as the process of discovery is in itself expensive, as we will see in Chapter 3, the assumption actually lowers the complexity of the problem.

We assume that the information being integrated has many text components and that its structure makes use of key / foreign key relationships. Since we use keyword searches to identify areas of the database that are of interest, we assume that there is a certain heterogeneity within the database itself. This arises fairly commonly when a database contains mostly human readable strings; which includes duty rosters, order shipment databases, and citation databases.

Databases with more homogeneous contents would include RNA sequence databanks, accounting ledgers and CAD part databases. Because these homogeneous databases have many reoccurring short strings, it would be difficult for us to locate the semantically correct one. Finally in many homogeneous databases, and especially in the case of accounting databases, the linkages between the relations are often maintained by an external application and not the relational model. In these cases, our approach will be unable to integrate the databases as the required information will not be present within the databases itself.

We expect that the database will have both indexes and primary/foreign keys and that the query engine will make use of these to process queries efficiently. However, in keeping with our assumption that the database client is unable to use schema information for integration decisions, we assume that this meta-data is also unavailable directly to the

database clients.

Therefore, we adopt a worst-case model of query costs and database operations. Intuitively, and from experimental results, we know that the major wall clock costs in database operations come from the traversal of relations and from transferring data between servers. Whenever possible, we constrain our queries so that any available index will be consulted by the query optimiser and so that the number of tuples returned will be restricted.

For example, when we perform free text queries of the database contents, not only do we restrict the number of returned tuples, but we also restrict the tuples to be processed to those that are known to be joined to other relations. Hence, not only do we restrict the amount of information returned, we also restrict the amount of information processed.

Interestingly, we also know that the marginal cost of retrieving an additional attribute from a tuple is negligible when compared with the total cost. Therefore, when retrieving results from the database we make an effort to retrieve as few tuples as possible while retrieving as many attributes from those tuples as possible.

From a purely practical point of view, we explicitly do not allow joins over a single relation between two instances of a same attribute, but we do support recursive joins where an attribute points back to another tuple of the same relation. Furthermore, for aesthetic reasons we represent strings with mixed case within the thesis, even through we assume that any character comparison is case-insensitive.

1.2.2 Thesis statement

Given an end-user owned, local reference database D^{local} with a single relation R^{local}, and a foreign source database D^{foreign} that contains related information of interest to the user, we wish to locate joined tuples within D^{foreign} that contain similar information to R^{local} and insert them into the appropriate attributes of D^{foreign}. Should additional information be located within the joined tuple, the option of inserting new attributes into R^{local} should be supported.

1.2.3 Proposed solution

The overall solution for the database integration problem detailed in this section is displayed in Algorithm 1. We make use initially of a free-form query provided by the end user

that is an example of the type of information that is required. A sample query is "blade runner harrison 1984" for integrating a movie database with his personal movie catalog.

Using this free-form query, we search the relations contained within the source and target databases and identify specific tuples that may be of interest to the user. This gives us a starting point for integrating the databases, as we now have similar instances within both databases that we can attempt to match and translate.

Hence, for the previous query we would find "harrison" in attribute person in the IMDB database and in attribute name in the MythTV database. We would be misled if trying to match the attributes based on schema information, since the IMDB database makes use of the name attribute for title information. By using a query, we are able to generate a small subset of similar instances that are likely to identify matching attributes and relations.

We should not immediately attempt to match and translate the information from one database to another, as this would incur large computational costs. Instead, we pursue an approach where we attempt to resolve the foreign key dependencies of the foreign database first (though without the knowledge of what attributes are actually keys). This is possible because we already have instances that are linked through an implied query relationship. Therefore, we can combinatorially search the foreign database for the specific attributes pairs that provide this foreign key relationship. Using this method, all of the query keywords can be used to identify the foreign key constraints that are most important to the information of interest to the user.

For example, with the shortened query "blade runner AND harrison" we can retrieve several instances for token "blade runner" on attribute table1.title and on attribute table2.name for "harrison". Searching for key relationships is then a simple combinatorial set of queries that attempt to retrieve tuples matching the query tokens across a potential join of two different attributes.

By using a similar process as above, we can also identify a projection of the information within the database. The two joined projections of the identified relations in both databases are then matched using the instances that have been located using the query tokens.

After some matching attributes have been identified in both database projects, we can next search for unmatched relations or attributes. Two basic steps are required: searching the foreign database for unmatched attributes contained within the local database and inserting those foreign attributes into the local database that contain new types of

Data: Based on the local database D^{local}, the foreign database D^{foreign} and a user query $\mathcal{Q}$.

foreach *Token* Q_i *in* $\mathcal{Q}$ **do**
  Pick best attribute in local database R^{local};
end
Find best tuple for $\mathcal{Q}$;
Sample $M-1$ more tuples from D^{local};
foreach R^{foreign} *in* D^{foreign} **do**
  foreach A^{foreign} *in* R^{foreign} **do**
    foreach Q_i *in* $\mathcal{Q}$ **do**
      Get similar instances to Q_i;
      Generate a NET() for Q_i;
    end
  end
end
while !*(*$\forall$ NET() *unjoinable)* **do**
  foreach *pair of* NET()_1 *and* NET()_2 **do**
    if *joinable* **then**
      Merge(NET()_1,NET()_2);
    end
  end
end
foreach $\mathcal{A}^{\text{local}} \in$ NET() in $\mathcal{R}^{\text{local}}$ **do**
  Translate $\mathcal{A}^{\text{local}}$ against R^{foreign} using NET();
end
foreach $\mathcal{A}^{\text{local}}! \in$ NET() in $\mathcal{R}^{\text{local}}$ **do**
  Match $\mathcal{A}^{\text{local}}$ against R^{foreign} using NET();
end

Algorithm 1: The integration process for our scenario.

information.

In the first step, we follow a method similar to the query method employed above. For a sample of records within the local database, we search the foreign database for relations or attributes containing the unknown attribute instances and that match the current sampled record. If necessary, new foreign keys to other relations are also searched.

In the second step, we search the local database for attribute instances matching the

new attribute and if no attribute is found, we insert a new attribute within a relation and update the dataset using the current relational data.

As this point, the linkages between both databases are satisfied and a series of relational queries are constructed to import the data from the foreign database to the local database.

1.3 Thesis Novelty

The novelty of this thesis lies in the algorithms that solve searching, matching and translation problems within and across relational databases without requiring pre-processing of the data or the indexing of the data on the foreign database.

Whereas in previous work these problems required privileged access to the databases and human intervention, our methods do not expect foreign database cooperation and require a minimum of human intervention.

To the best of our knowledge, this is the first work where the search for the joins that bind the relations is motivated by correctness instead of speed. Previous approaches relied on parameters to determine the reasonableness of joins, without concern for the problem of setting the parameters. In our approach, feasibility checking is based on obtaining consistent redundant information when applying a derived translation to the databases.

Previous integration methods required the use of schema information to locate both foreign keys matches and related information across databases. We make no use of schema information.

Finally, we provide methods by which the transformation of data from one database representation to another is determined from the data directly. No user intervention in terms of record alignment or correction is needed.

1.4 Thesis organisation and overview

In Chapter 2 we review the state of the art on data integration theory and some of the systems already proposed. The datasets used in this thesis both for performance evaluation and the integration scenario are described.

The organisation of the rest of the thesis follows the functional elements of the overall

process required to solve the integration scenario. For example, the search process of Algorithm 1 makes heavy use of the query and retrieval models that are covered in Chapter 3. Similarly, the creation of the query models for both searches and joins, require special database sampling methods that are also reviewed in that chapter. This is necessary in order for us to deal with extremely large and unknown databases.

Chapter 4 reviews the possible methods of performing joins between database tables with little information about the schema. We review previous methods of performing joins from a client's perspective. These methods are necessary to support both inter (matching) and intra-database joins.

After knowing the relationships between the tables of each database, we must find a means of translating the instances from one database to another. This is reviewed in Chapter 5 where we review previous work in that area and novel work on complex n to 1 translations.

Chapter 6 reviews the details of Algorithm 1 and of the experimental setup used to test its performance on real world data. We report on experiments with two different datasets and present conclusions regarding the algorithms. In all cases, these algorithms make conservative assumptions about the databases that are being utilised and about the amount of support the integration task is given. This implies that our experimental results are in most cases, conservative.

Chapter 7 concludes with a review of the novel work in this thesis and some future work required to solve still open problems.

Chapter 2

Previous Work

In this section we review the theoretical background of database integration. We first begin with the theoretical background and describe generic integration problems. Previous work in the areas is reviewed, and we finish by relating our own novel work against their work.

2.1 Theoretical background

The integration of several relational database sources has been modelled by Lenzerini [69], who defines a data integration system I between two databases as a triple $< G, S, M >$. G refers to the global schema of the integrated information, S is the source schema of a specific database, and M is the mapping between S and G. M provides the translation $q_s \rightarrow q_g$ that expresses a query from the global schema into the source schema, with both queries having their own languages and alphabets.

Database integration systems can be grouped into two large categories: global as view (GAV) and local as view (LAV), depending on the approach taken to the translation of the databases. In a global as view approach, the contents of the global schema is first defined and the contents of the source databases matched to them. In a local as view approach, the schema of the source databases is pushed to the global schema and integrated there.

Both methods are valid, with the advantage that the global as view method may produce the precise results required to meet the integration objectives. Local as view returns all information within the integrated database, which can be desirable when dealing with

previously unknown databases. Examples of local as view systems include the Information Manifold project at AT&T Bell Labs [63] and the XML system by Manolescu et al. [76]. Global as view systems include Garlic [19], COIN [47], and MOMIS [9, 11].

This model is commonly extended within the literature with two additional concepts, that of the wrapper and the mediator [82]. The term wrapper is normally used to refer to a software layer that handles the communications and the mechanical details of sending queries to the specific type or implementation of the databases. The mediator represents the conceptual process that implements and executes an integration strategy for one or more databases. This may be implemented as one or many mediators, and it is these conceptual blocks that capture the integration decision logic, including Lenzerini's mapping M.

The system proposed within this work takes a global as view approach to database integration, as we attempt to relate a data source against a previously existing database and take parts of the information only.

2.2 Integration Reasoning Approaches

A distinction in the systems and in the approaches is whether the problem of integration is taken as a deductive or an inductive one. Several engines take the position that database integration is a logic problem that requires finding a path linking all databases together. To do this, all type, instance and schema conversion functions are previously defined and the actual work of integration is a deductive logic exercise of aligning the correct logical conversions in the right order to achieve the desired integration function.

A second viewpoint, and the one used within this thesis, is that of inductive reasoning. We make no assumptions that conversion or matching functions exist and use the evidence produced by the databases to support a solution, even though it may not completely ensure its logical validity. It can be argued that deductive approaches to integration are best suited to integration problems occurring within the same software system, such as in the case of a federation in Figure 2.1, as the designs will have been formed from a core database dictionary. Furthermore, deductive systems tend to be tightly wedded to the database schema, which provides the starting point for reasoning to occur, while inductive systems rely on undeclared relationships inferred from the database values.

Examples of a deductive system would include the COIN system by Bressan et al. [15],

the SIMS system by Knoblock et al. [64], Ariadne by Arens et al. [7] and TRANSCM by Milo and Sagit [81]. All of the systems depend on accurate information about possible matchings, transformations and equivalences. Several systems, including SKAT by Mitra et al. [82], make use of formal knowledge bases to create their solutions. Conversely, Autoplex by Berlin and Motro [12] have a completely inductive solution where a Bayesian model is used to link various attributes, relations and tuples to a global databases. Another example is LSD by Doan et al. [31], which makes heavy use of schema names and instance values to produce its integration solutions.

2.3 Integration Patterns

Several generic integration problems can be distinguished by the relationships between the different databases being integrated. Figure 2.1 depicts a federated database, where several databases cooperate to exchange and serve data to various clients. While the data and schema being administered by each database may differ, the databases are assumed to be cooperating with each other. Furthermore, their design and purpose is assumed to be inherently compatible, in that they were produced by the same environment or designer. This is the simplest case in terms of the complexity of the search for a solution, as all of the information and meta-data are available within a common environment.

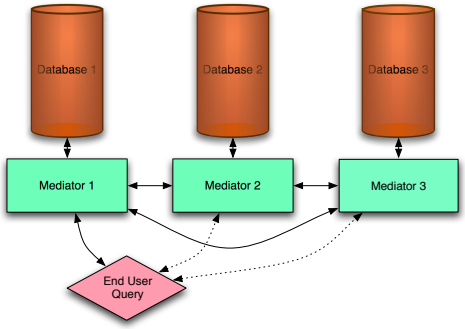

Figure 2.1: Basic federated data integration system.

A similar, but much more difficult, situation is depicted in Figure 2.2, where several databases are communicating on a opportunistic basis with no clear organisation. This

lack of organisation is reflected in the differences in the design of the databases and in that the integration solution is often a temporary one, such as described by Franconi et al. [40]. Because the integration tends to be for a short period of time and transient in its nature, the cost of finding new solutions is offset by the attrition of previous constraints imposed by disconnecting peer databases. Furthermore, the data gathered during other integration sessions allows the integration system to learn based on it previous experience by building a larger data dictionary, such as in LSD [31].

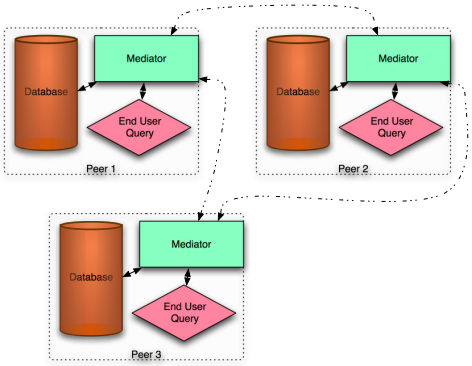

Figure 2.2: Basic peer-to-peer data integration system.

Figure 2.3 represents the canonical integration problem, where several databases are integrated under the same schema, as described by Templeton et al. [106]. The objective is to make most of the information available through the combined schema while insuring that the information is reconciled properly. This is one of the more difficult situations, as inconsistencies and conflicts can occur. This requires specialised code not only to handle schema mismatches, but also to handle complex integration problems that require information external to the databases. Take for example, the situation where one database represents historical information that was acquired at a certain date and another that contains fully time-stamped information. Integration logic is required to deal with the fact that the schemas are not only different, but that the integration can only be done if we know at what time the first database was created.

Figure 2.4 is the last case that we consider and the one that we attempt to solve generically in this thesis. We choose this scenario because it is a fundamental building block of all of the other scenarios, where a specific component of the foreign database

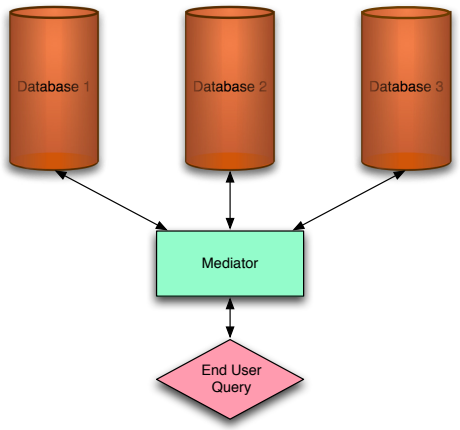

Figure 2.3: Basic data integration system.

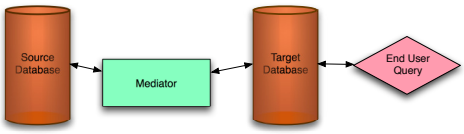

Figure 2.4: Basic load of information from one database to another.

must be integrated. All three of the previous scenarios must at some point go through a subprocess that mirrors the one in Figure 2.4.

This case is obviously less complex than Figure 2.3, in that we have only one foreign database to deal with. However, before we can deal with inter-database conflicts, we must first identify the contents and the mappings of the databases. Thus, one instance of sub-case Figure 2.4 must occur for each foreign database in Figure 2.3 before we can attempt final integration of all the systems.

This is also a sub-case of Figure 2.2 in that a process akin to Figure 2.4 must occur for each new peer. Most common database integration problems have a longer expected lifetime than that of Figure 2.2, where a rapid 'best-guess' solution is probably preferable to a perfect solution that is not obtained until after the peer has disconnected.

The peer-to-peer case has the advantage of accumulating a large store of meta-data due to the variety and frequency of integration tasks to be performed, which mitigates the speed requirement. For our purposes, we assume a boot-strap situation where no initial

data dictionary is available and where the necessary time can be allocated to a proper initial integration.

Finally, Figure 2.1 also contains some iterations of sub-case Figure 2.4 to select the proper translation and mapping functions from the data dictionary. This task is simplified by a complete data dictionary for the entire system. For our case, we assume that the data dictionary is unavailable, under the premise that a generalised solution to Figure 2.4, without meta-data, will also readily provide a good solution when provided with the full meta-data of Figure 2.1.

Hence, the case of Figure 2.4 is a fundamental base case that assumes the minimum in terms of meta-data availability. We present here a generalised solution to this sub-case which provides a generic component for database integration systems.

2.4 Research areas

Database integration research is a wide field, not only in its complexities, but in the specific subfields of research that it has engendered. Much of the early work concentrated on the interoperability between different systems and the underlying database algebras and transformation theories required to link different database types.

The first integration systems, such as those by Landers and Rosenberg [68] and Litwin [73] used a deductive approach to database integration. These concentrated on creating languages to express the various actions required to integrate the databases and the reasoning required to check the validity of the integration solution.

Some automation was attempted in creating systems that could reason through the integration process. These would accumulate the database design specifications and output mappings for the different schemas. Systems such as Hermes by Adali and Subrahmanian [1] and InfoMaster by Genesereth et al. [44] attempt to support the creation of solutions by finding a path satisfying all possible axioms to move the data from one database to another.

These approaches are based on a formal system of reasoning where the database is known and all of its behaviours explicitly declared. Hence, an attribute salary would be a positive decimal number implemented as a 2-byte integer whose value ranges from 30,000 to 100,000 and whose units represent Canadian dollars. A set of conversion functions that

match on this description convert any instance into another attribute, such as a salary in American dollars.

This level of information about a database is ontological in its details and such information is generally unavailable in most situations. Approaches by Levy et al. [71], ARTEMIS by Castano and Ferrara [21] and InfoQuilt by Sheth and Larson [103] attempt to create tools that convert one database's query to another by using outside ontologies. However, the problem of acquiring the background information for the required reasoning remained unaddressed.

Since most databases can at the very least enumerate their schemas and provide minimal meta-data, the idea of simple schema matching has been attempted. This approach recreates the equivalences required by earlier query rewriting systems by searching the various schemas for matching elements. The requirement here is to determine that $A_{yy}^{\text{foreign}} \longmapsto A_{x}^{\text{local}}$ so that an integration engine can deduce the conversion.

MOMIS [9, 11] builds a global schema based on the source schema and improves on the matching of schema name elements by using a Word-Net [79] database to attempt to match labels that are semantically related if not exact matches. The Clio project [80] uses both label matching and a Bayes classifier (using attribute instance values) to suggest potential matches to the end user. Cupid [75] also uses manual intervention by the user and leveraged the instances of attributes to attempt to match schemas. Madhavan et al. [75] claims that through their approach better performance was obtained than both the MOMIS system and DIKE [88] schema matching algorithm.

Embley and Xu [115] use both schema labels and attribute instances to generate schema matching, as well as a series of domain ontologies to match unknown strings. SEMINT by Li et al. [72] presents a matching system that uses neural networks to match elements from different schemas. This is done by acquiring 20 different metrics from both the data and meta-data and training a neural network with human generated examples. The DELTA system by Clifton et al. [26, 10] takes another approach by applying information retrieval methods on the meta-data itself. All available schema information and data-dictionary entries for each element is loaded within a text database and queried using the text tokens of another database to integrate. The elements whose meta-data score highest for each query was assumed to be a match for that element of the schema.

AutoMatch [13] uses an approach where a dictionary of all the database meta-data

has been manually acquired by the user. The dictionary is then re-used for any further integration project. The LSD project by Doan et al. [31] attempts to perform schema matching using the schema labels, but also makes use of specialised hand-created matchers that recognise schema elements based on statistical models of instances (q-grams or specific instances, such as a list of countries).

Similarity Flooding, an algorithm proposed by Melnik et al. [78], assigns a matching score between different schema elements in a bottom-up approach. Attributes are first scored for similarity, then the collection of all similarities for a certain relation is used to determine which other relation it is linked to. Finally, COMA by Aumueller et al. [8] makes use of a combination of schema labels, instance data, and ontological resources to identify the correct mappings. Meta-data from previous matchings is also kept in a knowledge database in an attempt to re-use the knowledge from one integration exercise to another. Finally, one of the more interesting projects is Autoplex, described by Berlin and Motro [12], which achieves simple translations and matches. The authors use several Bayesian methods on the actual instance data of the databases themselves. They then use the models not only to match schema elements across databases, but also to attempt to infer query constraints by comparing tuples sets from both databases under different query conditions.

The final research area that we review here is the question of the translation of the instances themselves. As previously stated, several assumptions have been previously made about the compatibility of the schema elements once matched. However, it may be unrealistic to assume that databases will represent the same piece of information in the same manner. One database might represent a person's name first-name first "Robert Warren", while another might opt to have last-name first "Warren, Robert". Note that we are concerned about the different possible representations of the same object. Other possibilities include the differences between time and date standards, part identification numbers and case normalisations.

We contrast this last area against the problem of data exchange where the objective is the materialisation of the instance. This research area is reviewed by Fagin et al. [33, 34] where specific instances are selected to be copied under different the constraints of the target schema. While this research area lends itself to very complex problems, such as the peer-to-peer scenario described by Fuxman et al. [41], the actual translation of the

instance representation is assumed to be compatible.

The Clio system described by Miller [80] has limited support for the description of translation tables. For example, one can define a translation table converting human-readable months to numerical ones: {Jan ⇒ 01, Feb ⇒ 02, Mar ⇒ 04, ..., Dec ⇒ 12}. The IMAP system by Dhamankar et al. [30] takes a much more detailed look at the problem, especially at the issue of learning mathematical conversions. By using equation learning methods, they propose a method whereby the mathematical equations relating different attributes are deduced using tuple values. The end result is that relationships such as Cost = UnitCost * NumberOfUnits are recovered from the data automatically, even if at a high computational cost. They also suggest that "format learners" should be constructed to infer differences in the representation of instances using the data directly from multiple databases.

2.5 Relationship of proposed approach to previous work

This thesis concerns itself with the retrieval and translation of a subset of a foreign database into the format of another local database. This is an inductive solution with a "global as view" approach to the problem of relating outside information to a local database. Only the information required by the application environment is integrated. Furthermore, when we perform our data matching we make no use of schema information as do COMA [8] , LSD [31] and IMAP [30]; we actively search for joins to link the information as Berlin and Motro [12] suggest. Finally, the instance data is translated from the foreign representation to the local one without user intervention in a manner that was suggested by Dhamankar et al. [30]. To the best of our knowledge a method that translates generic instances, locates and verifies potential joins within a database, and performs data matching based on instance data only without user input is novel. The only end-user involvement is the entry of a free-form query that serves to identify what specific type of information is required of the integration.

Chapter 3

Sampling and Retrieval Models for Databases

If you know the size of the database, it is not a very large database.

A discovery component is often part of the database integration process. We may not be completely aware of the information contained within a database, or we may not know where in the database the data of interest is located. In effect, we require two basic procedures: a means of searching the database for information of interest and a means of discovering the proper joins between the relations to obtain a single, possibly compound item of information, for multiple relations of interest.

A complication to these tasks is our lack of knowledge concerning the contents and size of the database. While the schema can be used to reference the various relations within the database, we are without information as to their contents or the number of tuples within them. This complicates our search approach in that size information is a valuable piece of information to plan an effective and efficient search.

The situation depicted in Figure 3.1 is typical of the architecture that we face in several integration situations. The foreign database is accessible through a computer network and its contents unknown. We cannot simply copy over all of the information and analyse it locally; we must make use of the query facilities available.

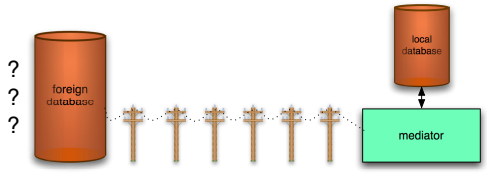

Figure 3.1: From a set of known instances, we wish to retrieve instances from an analogous column with similar instances.

In this chapter, we review potential sampling and retrieval solutions that can be used to solve the problem of the unknown database. We present a novel retrieval solution implementable within relational databases that can also double as a means of suggesting potential joins. Evaluating the joins is the topic of the next section. We close the chapter by providing a performance comparison of the novel method with previous methods.

3.1 Previous work

Here we begin by reviewing previous work done in the retrieval of inexact instances from relational databases and client-side sampling methods. Both of these areas are different from traditional database methods, in that relational database researchers have historically assumed an exact instance match paradigm as well as server side sampling.

3.1.1 Imprecise retrieval

Database retrieval has been investigated in both the traditional database area and in the information retrieval area, differing mostly in the query answering model used. Classical databases tend towards precise queries retrieving specific instances, whereas information retrieval prefers top-k ranked retrievals based on a likelihood estimate.

Given a topic T and a set of documents D, an IR system returns an ordered subset $S = s_1 s_2 \dots s_n$ of D, ranked according to an estimate of the likelihood of relevancy of the document to T. In this thesis we use the generally accepted Average Precision (Equation 3.1) metric as defined by Cormack and Lynam [27]. By averaging the Average Precision metric over several topics, we obtain a Mean Average Precision (MAP) score that we can use to benchmark different retrieval systems.

$$AP = \sum_{k=1}^{|S|} rel(s_k) * \text{Precision}(k)/R \tag{3.1}$$

$$\text{Precision}(k) = \sum_{i=1}^{k} rel(s_i)/k \tag{3.2}$$

$$R = \sum_{d_i \in D} rel(d) \tag{3.3}$$

$$rel(d) = \begin{cases} 1 \text{ if document } d \text{ is relevant to } T \\ 0 \text{ otherwise} \end{cases} \tag{3.4}$$

In our work we require both of these query models. Any eventual integration solution will obviously need to respect the integrity constraints of a classical relational database. But to search for the proper integration solution we need to find objects that are represented differently in multiple databases.

As early as SQL99, there has been support for the use of regular expressions to retrieve certain instances from text attributes. Furthermore, many of the modern systems have an indexing facility for text. However, the purpose of these indexes is the rapid search of the data using the same regular expression facilities as an un-indexed attribute would use. Furthermore, these are not necessarily provided for the particular database in use or not available to all users. For this reason, we require a method generalisable to most databases' query engines. The indexing that we review here allows users to search the text using similarity metrics that aren't supported directly in most systems.

The complication lies in the generation of a query that will search for instances within a certain similarity distance. Determining what model should be used to make a decision on acceptable dissimilarity is a challenge.

A simple example is the use of lower and upper case characters within databases. It is reasonable that some databases would represent names in a variety of formats, for example all uppercase "ROBERT", all lowercase "robert" or in mixed case "Robert". By inspection, the retrieval would be very simple for a human being, while requiring processing for a machine. This particular retrieval problem can be solved by using one of the database case normalisation functions as part of the query.

In traditional relational database theory, we query relations based on exact matches; an instance matches another or it does not. In the context of data-driven integration, the instances being searched for are likely to be represented in different ways. This means that we require a way by which our queries will tolerate inexact matches between instance values.

Because the foreign database is mostly unknown to us, we require a means of performing these queries in a way that can be done on the database itself. It would be preferable to have all of the data within a general programming environment where we could manipulate the data as we wish. But the unknown scale of the database puts us into a situation where the data transfer can be prohibitively expensive.

The canonical solution for most string comparison models is the use of q-grams [108]. Short substrings of length q extracted from two strings are matched against each other. As an example, the string "possible", contains five 4-grams, namely {"poss", "ossi", "ssib", "sibl" and "ible", and so a string s of length $|s|$ contains $|s| - q + 1$ q-grams. We then compare strings x and y from attributes X and Y according to the number of distinct q-grams that are shared by each string, in the manner of Equation 3.5.

$$\text{ScorePair}(a, b) = \sum_{\alpha=1}^{\text{distinct}(|s|-q+1)} \begin{cases} 1, & \text{if instance } x \text{ of } X \text{ and instance } y \\ & \text{of } Y \text{ share } q\text{-gram}_\alpha \\ 0, & \text{if not.} \end{cases} \tag{3.5}$$

This allows us to identify string pairs that have some sub-strings in common and hence have a high likelihood of being similar instances.

However, this still does not normalise the relative frequencies of the q-grams. Long strings containing several commonly occurring q-grams would be ranked higher than shorter stings with rare and relevant q-grams. Hence, we assign weights to each q-gram according to the frequency with which it appears in both the individual string and string collection.

Equation 3.6 represents the Term Frequency - Inverse Document Frequency formula (TF-IDF, see Salton et al. [99]) for calculating a weight for an individual q-gram within a string s with a string collection (attribute) S: $\text{w}_{s\alpha}$ is the weight assigned to the αth q-gram of string s, where the first term $\frac{|q-\text{gram}_\alpha \in s|}{|s|-q+1}$ represents the frequency of gram α within the specific string s and $\frac{|S|}{|q-\text{gram}_\alpha \in S|}$ represents the number of strings within the set over the

number of strings that contain the αth q-gram.

Equation 3.7 then represents a simple weighed scoring function for a pair of values from a and b. As before with Equation 3.5, we iterate through the list of possible q-gram within both strings, but weight each gram based on their score for both strings. This prevents a q-gram that is too prevalent within either string set from skewing the results.

$$\mathrm{w}_{s\alpha} = \frac{|q{-}\mathrm{gram}_\alpha \in s|}{|s| - q + 1} * \log \frac{|S|}{|q{-}\mathrm{gram}_\alpha \in S|} \tag{3.6}$$

$$\mathrm{ScorePair}(a, b) = \sum_{\alpha=1}^{\mathrm{distinct}(|s|-q+1)} \begin{cases} \mathrm{w}_{x\alpha} * \mathrm{w}_{y\alpha}, & \text{if instance } x \text{ of } X \text{ and instance } y \\ & \text{of } Y \text{ share } q\text{-gram}_\alpha \\ 0, & \text{if not.} \end{cases} \tag{3.7}$$

The canonical use is to implement an information retrieval engine on top of the database using a q-gram retrieval model. This approach has already been implemented within an SQL database by several researchers for inexact string matching applications. Koudas et al. [67], Chaudhuri et al. [23] and Gravano et al. [51] all use variations of this approach to match similar records using TF-IDF and cosine similarity [100]. All of these approaches require the generation of index tables.

The approach is reliable but suffers from two major drawbacks: it is computationally expensive in time and storage and does not deal well with situations where the q-gram size is poorly set.

Using a TF-IDF model requires the storage of a gram weight index on the server, with an approximate creation cost of $t + t * (w - q + 1) + \frac{t^2*(w-q+1)^2}{m}$, where t is the number of tuples within the relation, w is the average character width of the attribute being indexed and m is the number of unique grams present within the attribute.

The total storage cost of the index is about $t * m_{avg}$ where m_{avg} is the average number of unique grams per instance within the relation. Koudas et al. [67] were able to amortise some of this cost of building and storing the index by setting minimal score thresholds for q-gram retention. Similarly, Mazeika et al. [77] used similarities within the strings and q-grams to reduce the index size while providing an estimate of the selectivity of the query.

Finally, there exists no simple way of updating the index as the data changes. The weights required to calculate the scores are based on collection and document wide averages that must be recalculated for each update and doing this over an existing database system is non-trivial.

3.1.2 Database statistical sampling

Sampling is heavily used by most database management systems to optimise their internal processing, with good surveys provided by Gibbons et al. [46] and Olken and Rotem [86, 85]. The research is primarily one that is oriented to server-side query processing and optimisation.

Most of these methods rely on an intimate knowledge of the physical layout that the database creates and of the storage devices that are in use. Because of their close proximity to devices, both the query optimisers and the sampling methods that they use function in terms of blocks of storage.

A mapping layer translates the needs of the query engine to the storage layer. However, since a relation behaves as linear storage when used through the query engine, we cannot use the sampling methods meant for query optimisers that assume a block view of the problem.

The earliest known work in the database application area is by Jones [60], who described algorithms for the random sampling of records from magnetic tape using random length intervals. With the advent of random-access devices, the possibilities for sophisticated analyses have increased as storage is no longer a linear access device.

Chaudaury et al. [25, 24] covered some of the possibilities of block sampling in terms of optimising how many tuples should be sampled at the least cost. Utilising the block nature of the storage as a iteration size, they approached the problem of sampling by incrementally adding more data to a histogram with an error measure. The process was repeated until the error converged to a stable value.

Their work was in the context of query optimisation within Microsoft SQL Server and assumes that equi-height histograms are being generated for arbitrary precision numerical values. It also assumes the possibility of addressing direct disk blocks. Hence they use this to their advantage by pre-computing the amount of data that each block can contribute

to the histogram and discarding it if its values are too similar. This limits its usefulness for database integration work if the values are retrieved from a remote database without block addressability and the values required for inter-database joins are strings.

They also revisit some work done by Haas et al. [52] on the estimation of unique values within a table. This is useful in that it permits the prediction of the size of a join [83] or of the number of possible unseen entries left within the relation. Much of this work rests on previous research on the catch and release of animals in the wild: how long should one keep trapping animals before concluding that another species would be identified. Such methods have been provided by Fisher et al. [37] in non-parametric models and by Good and Toulmin [50] for parametric models. Chaudhuri et al. [25, 22] provide a well performing estimator for the number of distinct values within an attribute, with a slightly better error rate than the one by Haas et al. [52]. Bunge and Fitzpatrick [18] review several of these methods in the context of distinct value prediction in relational databases.

Straightforward methods of sampling also include equidistant sampling where a certain percentage of tuples are selected from a relation at equi-distant location. Similarly, random position sampling would select a certain percentage of tuples at uniformly distributed locations within the table.

Most recently, Gravano et al. [51] compared the performance of equidistant sampling versus random position sampling and found the two methods to be equivalent. The equidistant sample is sometimes preferable to a random position sample, as we do not need to pre-compute the random lengths in order to sample the values in a single pass.

Another sampling method that has been heavily used in the area of limited-length data streams or sequences is Reservoir Sampling. In this approach a limited number of rows are retained while traversing the dataset and this reservoir of rows is taken to be a representative sample of the dataset.

First presented by Fan et al. [35], the algorithm is attractive because of its limited memory footprint: a set sized "reservoir" of values is created to hold the sample. As we iterate through the attributes values, we use a random variable to determine whether this particular instance should be inserted into the reservoir.

This research was continued by Vitter [111, 110] who improved on the process by adding the use of 'skipping' operations where blocks of rows would not be inserted into the reservoir. This speeds up the sampling process by using seek operations instead of

merely read operations in databases. The intuition behind the skipping algorithm is that we must calculate the number of times that we are likely not to select a value for any of the slots within the reservoirs. In this manner we use an expansion of a number of Bernoulli trials to calculate the seek distance. Similarly, by using a random variable, we calculate the number of successful Bernoulli trials.

These processes enable us to sample the database in a scalable way. However, they all require us to know the size of the database before the sampling process can begin.

Park et al. [89] revisited this problem with the objective of removing the constraint that the size of the dataset should be known a-priori. Instead, each new instance can be repeatedly assigned to any given slot with the same probability. Hence for any sampled value, and a reservoir size of m, the probability that a value will be inserted into the reservoir is determined by m Bernoulli trials. For each success, a value is removed from the reservoir as in the original method.

Johnson et al. [59] implement this method as a specialised operator within a relational database system for extremely large databases. Additional work by Park et al. [89], similar to Vitter's work, suggests also using the 'anytime' algorithm with skipping, which can result in a speedup, as not every value needs to be transferred to the sampling algorithm. Furthermore, since it does not need to know the length of the table, we can have the sampling method run in the background of our integration process, thus ensuring that the best information about the relation content so far is available. Ironically, this creates another problem, in that we are unsure of the validity of the current reservoir contents unless a significant amount of information has been processed. How much data is significant is unclear when the dataset is of unknown of size.

3.2 Research Problem

The work done in this area must be adapted before it can be used within our own application domain. Specifically, some scalability, cost and fundamental assumptions about the data are invalid without a-priori information from the database. We review here some of these issues.

3.2.1 Inexact retrieval within large relations

In the case of retrieval engines implemented over a relational database, we are limited in what we can implement within the DBMS as a similarity retrieval function. The traditional information retrieval approach uses Term Frequency - Inverse Document Frequency (TF-IDF) approach, which in our integration model will require the creation of a q-gram index table by the client. Even if we are in a position to avoid the transmission cost required to form the q-grams from the strings, there remains a large processing and storage cost to this method.

Furthermore, we may not be in a position to create temporary tables. Even if the database is not an adversarial one, it may be non-cooperative and prevent us from creating server-side temporary tables. Similarly, even if we have the option of creating temporary tables, we would prefer not to index some relations altogether if we judge them to be non-relevant or too small to warrant indexing. Thus, it would be preferable to have a retrieval method that would not require an index or pre-processing and yet be applicable on any arbitrary relation within the database.

Another problem is that setting the q parameter, or the length of the grams, must be decided at the onset of the retrieval operation. There exists a large computational advantage to increasing the size of grams, as this decreases the number of grams generated for each string instance. This comes at the price of a decrease in the recall of the instance retrieval function, as large sequences of characters will need to match exactly for them to be considered similar.

Another concern is the q-gram model that is used to estimate the similarity of strings, in that it may not account for all possible transformations. Consider time-stamps, where the information that is valuable within it is in sequences of characters (hours, minutes, day, year, ...) separated by constant separators. However the separators and the order of the sequences used might differ between databases. Should we use tri-grams to index such a string, no gram value would ever be the same (e.g., ":30" would not match "-30").

Another approach that is possible is the concurrent use of multi-length grams from the same data [108]. Intuitively, we know that not all parts of a string have the same importance. Space, or blank, characters for example, convey no information save to act as separators for different pieces of information, such as words in some human languages.

For our purposes, it is not always clear where we can segment an attribute instance for maximum retrieval efficiency. The problem is similar to the one of text segmentation, where words from a sentence stream are segmented. Most of these methods, such as the one proposed by Peng and Schuurmans [91], depend on a lexicon or dictionary being available on which machine learning methods can operate; these are not available in our case. Similarly, Goldsmith [49] does suggest a gram oriented probabilistic method for making gram selection, and Ge et al. [43] present a dynamic programming solution that is based on attempts to segment characters in a stream. Unfortunately, these approaches are oriented towards large strings, and they do not apply well to our problem area.

3.2.2 Sampling from a set of unknown relations

We wish to explore a series of database relations about which we know very little. The intent is to locate information that the user has specified as the integration starting point and to infer the structure of the database from the data.

With the schema providing a means of referencing itself as relations, we need a method of analysing each relation for its contents in a fashion that is scalable and useful to our integration objective. Previously, Dasu et al. [28] suggested that "sketches" of each attribute and relation should be constructed to allow for them to be compared to each other or a query. This requires the traversal of each attribute as a histogram or content (q-gram) sketch is built from the collected instances. Similarly, Rowe [98] presented an idea for a standardised statistical profile that would be computed for all attributes within the database. Since this was meant to improve the performance of optimisers on the database server itself, little thought was given to its use from a client perspective.

Similarly, the SEMINT system by Li et al. [72] automatically collects statistics from every attribute within every relation in a database before attempting to match schemas. This requires at a minimum $J * |\mathcal{A}_j^{\text{foreign}}|$ relation traversals to acquire only the statistics about the attributes, without any indication about its content. Thus, the computational cost imposed on the system is large even before the integration process begins.

A problem with most sampling approaches is that they require the traversal of a single relation multiple times, as in Figure 3.3, in order to sample every single attribute individually. This is a high computational load on the database as the relation is traversed from

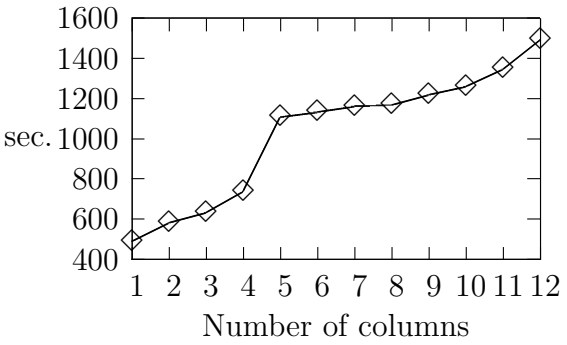

Figure 3.2: Averaged wall clock retrieval time for different numbers of attributes concurrently projected from a test relation.

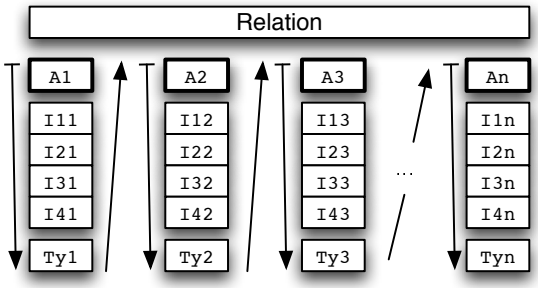

Figure 3.3: Current sampling methods involve traversing the same relation several times in order to sample each individual attributes.

top to bottom several times. Hence a possibility would be the sampling of the relation itself and then the generation of histograms on the individual attributes.

This is observable in Figure 3.2 where the time required to retrieve all the tuples are plotted against the number of attributes retrieved. The relation contained a million tuples with each attribute containing 256 characters on an instance of the Postgresql relational database system version 7.4.1 running on a Sunfire v880 750MHz machine. Notice that while the marginal cost of retrieving each additional attribute does rise, it still is negligible when compared to running individual queries for each attribute. The 'step' in processing time located at the fifth added column is the result of buffer sizing effects [117]. Thus it is advisable to retrieve multiple attributes on an opportunistic basis when querying an unknown database.

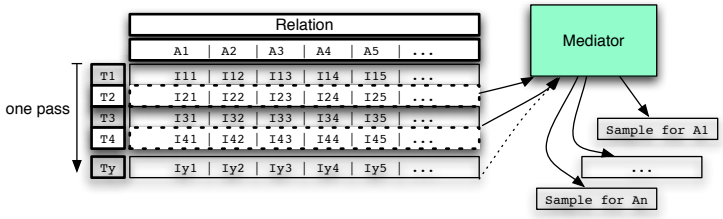

Figure 3.4: One pass reduces the time cost as the relation is traversed only once.

Similarly, all retrieval methods follow an attribute-by-attribute model that requires multiple passes through a relation. We would prefer a model that would allow us to index or retrieve from a relation for all attributes at one time, as in Figure 3.4. A possibility is to use the CASE SQL construct as in Query 3.1. This "un-rolling" of the relation permits the selection of the highest scoring attribute from the tuple results into a final column that is then sorted by the aggregate function, in a manner not-unlike the PIVOT operator used by Fletcher [38].

Query 3.1 Querying all attributes from the relation

```
select title, name, kosinov(query, name) as namescore,
kosinov(query, title) as titlescore,
case when titlescore > namescore
then titlescore else namescore end as mainscore
from table order by mainscore desc limit 10
```

Since any computation involving one row is negligible when compared with the cost of retrieving the row from the database, we propose here to pre-compute all of the similarity scores of the tuple attributes and pre-sort the attribute of the tuple before the results are fed to the query aggregate functions. However, it was found that in many cases while this was a theoretically sound idea, database optimisers such as the one in use in the 7.3.1 version of Postgresql would simplify this query into sequential operations which would actually slow the results instead of speeding them up. While this strategy may no doubt function for other implementations, we thought that its was not sufficiently generalisable for inclusion in our system.

Table 3.1 represents all of the sampling methods previously reviewed in Section 3.1.2, and whether they require the size of the relation a-priori and any problems with the method.

The concern with knowing the size of a relation is that in many cases acquiring this information requires traversing the relation itself. As previously stated, the database concerns itself with storage blocks while the query interface concerns itself in tuples; the database may not record the number of tuples within a relation. Storage blocks can contain deleted tuples that have not been purged. While it may be reasonable for the optimiser to estimate tuple counts based on blocks counts, the query engine requires an accurate count to answer a query. Hence, if a current count is not stored just acquiring a $G_{\text{count}(*)}$ of all tuples will

Method	Problem with use
Read the entire relation	Un-scalable
Random (interval) sampling [60, 51]	Un-scalable, requires size
Reservoir sampling without replacement [35, 89]	Un-scalable, requires size
Reservoir sampling with replacement [111, 110]	Scalable (1 pass), requires size
Reservoir sampling with replacement (stream)[89]	Result validity unknown
Histogram-error driven sampling [25, 24]	Undefined results in sorted attributes
Prediction of unseen distinct value [37, 22, 50, 52]	Parameter setting and result validity.
Query based sampling [104]	Need to select appropriate query.

Table 3.1: Comparison of various sampling strategies for attributes within a relation.

force the database to read all of the pages allocated to that relation.

In practice, this means that any relation that we attempt to sample is likely to trigger two traversals of the relation: the first to acquire the tuple count of the relation and the second to sample the tuples.

Hence, we would like to avoid any solution that requires obtaining the size of the relation first, which rules out Random sampling, Reservoir without replacement, and Reservoir sampling with replacement. One approach is to track an error metric, such as Kullback-Leibler divergence, as a histogram of sampled values is constructed and stop the sampling as the error converges.

However, this fails on sorted attributes; the histogram may immediately converge for the first few instance values and immediately terminate. It is possible to add a minimum bootstrap to the number of instances sampled, the problem then becomes one of parameter setting against an unknown length relation. In fact, random sampling for a sorted attribute requires random access, which is not available through most interfaces, including ODBC and JDBC.

The streamed version of Reservoir sampling with replacement is a possibility, as it is meant as an anytime algorithm that does not require the relation length to function. The problem is that it is unclear how long to wait (measured time, number of instances, reservoir, ...) before we can consider the sample accurate and that randomly distributed instances as assumed. This creates a situation similar to that of the histogram-error method, in that we now need to set a parameter blindly. A possible work-around could be the use of an estimator for the number of distinct values remaining within the relation. Some of

the methods by Fisher et al. [37], Charikar et al. [22], Good and Toulmin. [50] and Haas et al. [52] lend themselves to this estimation without requiring the length of the relation. It remains unclear, however, how we could transform this information into a stopping condition.

One of the conclusions drawn from retrieval experiments is that the sampling does not need to be statistically unbiased. Because our end-goal is to link the information from the relation into another database, the only tuples which we really wish to sample are those that can support a linkage with another relation in a database. Hence, what we are really concerned about in our sampling is whether the sample is *relevant*. This notion is hinted at by Popa et al. [92, 93] where they examine data chasing and by Fletcher [38] who look for critical instances that must be obtained in order for the operation to succeed.

3.3 Adopted Approach

Based on the issues identified in the previous section, we propose here a unified method both to retrieve inexact matched instances and to provide limited support to the problem of locating joins within a relational database.

We may not be in a position to create temporary tables in order to create indexes, or we might not wish to create an index for a relation that may only be queried a single time. Thus, it would be preferable to have a retrieval method that operates on any arbitrary table without an index or pre-processing. We do know that the cost of processing a query lies in the traversal of the relation itself. It would therefore be in our interest to make use of a retrieval model where the similarity is computed in-line with the reading of database pages, hence directly within the query itself.

From our perspective, we only have access to the data through a standardised query interface that may or may not have caching and optimisation built into the interface. Furthermore, recall that the database is large and cannot simply be loaded into main memory. We therefore sample values from the database whenever possible to reduce the amount of information being processed.

Hence, we make use of a string similarity function as a retrieval scoring function that does not require the presence of an index to compute q-gram weights. The similarity function (3.8) proposed by Kosinov [65] compares the number of common q-grams normalised

to the number of unique grams (C_{String}) within each string.

$$S = \frac{q * C_{common}}{C_{String1} + C_{String2}} \tag{3.8}$$

We modified the Kosinov similarity measure by replacing the gram normalisation term $C_{String2}$ with the length of the string. This inserts a bias in the formula towards very long strings with redundant information. However, it also allows us to insert the formula directly within an SQL statement without the need for an additional join to count the number of unique grams within the target attribute instance. Equation 3.9 can then be directly implemented as an in-line function to a simple ranking query.

$$\widehat{S} = \frac{q * C_{common}}{C_{String1} + \text{length}(String2)} \tag{3.9}$$

This relies on an assumption that q-grams do not repeat often within the strings being queried within databases, as we substituted the count of unique q-grams with the length of the string. Figure 3.5 plots the histogram of the distance between repetition of tri-grams in an English text. The average distance between repeated tri-grams is 32 characters, which gives us a certain amount of confidence in our assumption for short textual strings.

The bias needs to be taken into account when making use of the results in the integration process, however some specific types of data with small repeating sequences, such as DNA data e.g.: "CCAAGTACCCAAGTAC", may prove very difficult to handle, but this is not the case here according to our data assumptions in Section 1.2.1.

The power of this retrieval model also lies in our ability to introduce constraints at retrieval time without the requirement that any such feature be previously indexed. For example, we are able to select a specific subset of tuples from a relation with specific values and search them for specific substrings at the same time.

Using some mathematical properties of basic SQL functions we are able to implement this query into SQL as in Query 3.2. We can easily limit the number of return results to a top-k listing, by sorting the results by their computed score. Not all of the instances retrieved will be relevant to the query, and the value of k is arbitrary. By combining the results of multiple queries, we are able to create a subset of the data at no additional cost.

A similar method was used by Si and Callan [104] to explore online web databases by

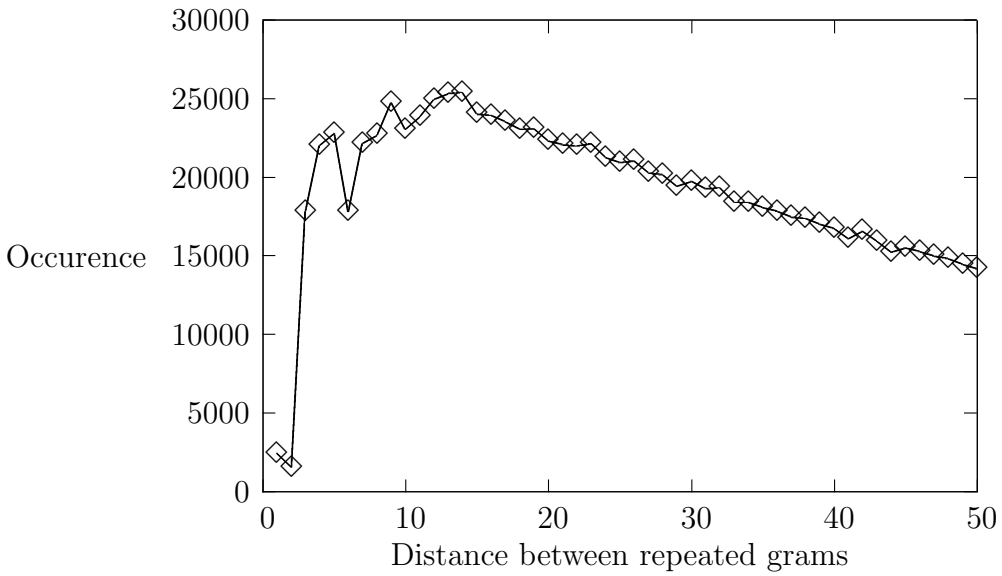

Figure 3.5: Histogram of the distance in characters between each repetition of a q-gram within War and Peace [107].

selecting grams to gradually map a web-site's data incrementally. Bruno et al. [17] used a similar method driven by query constraints to explore a database. Here the query tokens have previously been chosen by the user through the initial search query, and in Chapter 4 we explain how to make use of the retrieved instance as a sample of the attributes to identify possible joins within the database.

3.3.1 Dealing with query terms containing separators

We previously reviewed in Section 3.2.1 how choosing a length of q-gram would affect the performance of the retrieval. The concern is that when we search for an specific instance, we may have the wrong q-gram length and have no way of breaking down the data into its basic components.

While there are no current means of optimising the value of q for a free-form query term, there is a means of doing so when using an attribute instance as a query term. This is because we can easily identify patterns within a series of instances from an attribute, as opposed to a one-off term from a user query.

Query 3.2 Querying the title attributes from the program relation. Note that the leading 0.0 is necessary for the Postgresql query engine.

```
select title, 3*(0.0 + (1 - pow(0, position('bla' in title)))
+ (1 - pow(0, position('lad' in title)))
+ (1 - pow(0, position('ade' in title)))
+ (1 - pow(0, position('de ' in title)))
+ ...
+ (1 - pow(0, position('ner' in title))))
/ (12+ char_length(title)) as score from mythtv.program
order by score desc limit 1000;
```

This is done by taking a histogram of the all non-alphanumeric characters within the target column against all potential character positions. However, in order to be able to handle strings of variable length, we use relative positions allowing for as many positions as there are characters in the average length of the instances within the target column. For example, if the average instance length were 5, we would compute 5 relative positions, and if the current instance length were 10, we would retrieve the 4^{th} character when generating a histogram for relative position 2. (Note that this simplifies to absolute positions when a column is of fixed width.)

For example, the histogram in Figure 3.6 plots the occurrence frequencies of potential separators for a set of first and last names separated by a comma and space in the manner of "Doe, John". Since the rounded average length for the column is 15 characters, we plot the histogram for relative positions 1 through 15.

From the histogram, we can see that there are many comma and space characters in the middle of the instances. We now need an algorithmic way to select which of these candidate separators and locations are actually valid for all column instances.

A candidate separator at some location is invalid if there is at least one instance that does not include it in that position. For a fixed column width, it would be sufficient to set a threshold to the number of instances within the column and simply select the characters and positions that score above it within the histogram. However, for variable width columns, we must verify the separator template, as it is possible for artifacts of the data to generate an incorrect separator format. We therefore start by examining the most common separator/position pairs, and testing whether a template specifying those

Data: A set of database instances $\mathcal{RHS}$ representative of an attributes A^{foreign} of a relation R^{foreign}.

```
1  Histogram(Location, Separator) ← 0;
2  AvgLength = Avg(Length(RHS));
3  Grams ← 0;
4  for x = 1 to AvgLength do
5  |  foreach Separator character s in {−, /, _, :, ., ..., } do
6  |  |  foreach Instance RHS of RHS do
7  |  |  |  if charAt(x/AvgLength*Length(RHS))==s then  Histogram(x, s)++;
8  |  |  end
9  |  end
10 end
11 Threshold = max(∀Histogram);
12 SearchKey = '%';
13 TestSearchKey ← SearchKey;
14 repeat
15 |  SearchKey ← TestSearchKey;
16 |  for x = 1 to AvgLength do
17 |  |  foreach Separator character s in {−, /, _, :, ., ..., } do
18 |  |  |  if Histogram(x, s) > Threshold then
19 |  |  |  |  TestSearchKey = TestSearchKey + s;
20 |  |  |  else
21 |  |  |  |  TestSearchKey = TestSearchKey + '%';
22 |  |  |  end
23 |  |  end
24 |  end
25 |  Threshold−−;
26 until G_count(*) σ_{A^foreign !LIKE TestSearchKey}(R^foreign) > 0;
27 return SearchKey: A representation of the separator pattern;
```

Algorithm 2: FindSeparators($\mathcal{RHS}$) returns a representation of the layout mask used by a certain attribute for its instances.

separators in those positions matches all the instances. If so, we augment the template to include the next most common separator-location pairs and continue until a candidate template no longer matches all instances.

Algorithm 2 encodes the building of the histograms followed by the search for the appro-

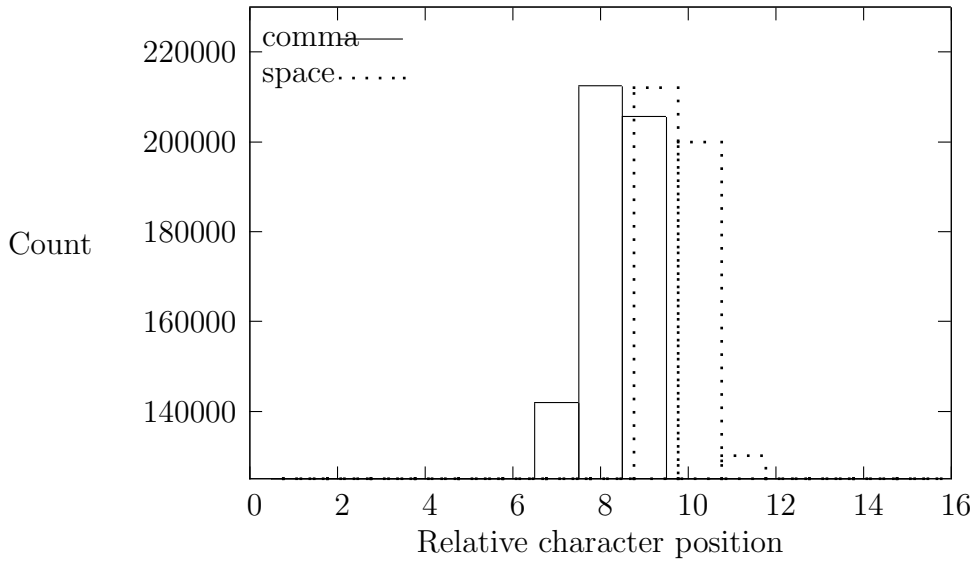

Figure 3.6: Histogram of possible separators and their locations for a series of separated first and last names, which would indicate that the comma should deliminate separate n-grams.

priate separator locations by repeatedly lowering a threshold controlling which separator-location pairs to include. Using this algorithm, we are able to recover the locations at which tokenisation should occur. For example, the attribute represented by Figure 3.6 would return a mask similar to "%, %" which would split the sample instance "Doe, John" into the tokens "Doe" and "John". These two tokens would then be split if they are longer that the chosen q-gram value. Thus for a q value of three, the resulting tokens would be "Doe", "Joh" and "ohn".

Compared with a normal tri-gram tokenisation ("Doe", "oe,", "e, ", ", J", "Joh"and "ohn"), this method therefore returns fewer search tokens while avoiding unmatchable tokens. Because this method forces the tokenisation of the query term at arbitrary locations, it is difficult to use with indexes as these will not have been prepared for the specific grams, we may end up with bi-grams or singletons, that will have been indexed. Therefore, we are only able to make use of this method with non-indexing methods.

3.4 Experimental Results

In this section we present some experimental performance results for our proposed method using synthesised data as well as the IMDB database [29] and a dump of a MythTV [54] database instance.

Data Set	Database A	T'	Database B	# tuples
ISO-8601 Long hand	2038-01-19 03:14:07	⇒	2038-01-19T03:14:07+00:00	
ISO-8601 Short Date	2026-10-15 05:19:55	⇒	05-19-55	
ISO-8601 Short Time	1990-05-25 10:17:42	⇒	10:17:42	1,516,524
Canadian Short Date	1984-06-05 19:01:26	⇒	1984-Jun-05	
US Short Date	1987-09-22 05:10:02	⇒	09/22/87	
Unix login name	warren	⇒	rhwarren	8,336
Last name versus full name	warren	⇒	warren, robert	701,466
Random person in IMDB	Harrison Ford	⇒	*full database*	2,075,695
Random title in IMDB	Blade Runner	⇒	*full database*	896,117
Random person in MythTV	Harrison Ford	⇒	*full database*	39,418
Random title in MythTV	Blade Runner	⇒	*full database*	25,570

Table 3.2: The different representations of the same instance data used to benchmark the performance of the retrieval methods.

Table 3.2 presents some of the synthetic datasets that we use to benchmark the ability to retrieve different representations of the same instance data. The first 5 datasets are date and time representations, as described by Wolf and Wicksteed [114], which are often problematic in databases. The next two datasets are textual attributes of names that reference both Unix login names and long hand versions of the same name. Note that not all of these transformations are complete, some of them are either lossy or are incomplete, as matching across databases must occasionally occur with incomplete information.

To perform retrieval on either the MythTV or IMDB databases, we randomly choose a string from the relevant attribute and randomly corrupt 10% of the characters within the string with a random character. We do this to provide an element of transformation to the retrieval function that mimics integrating different representation of the same data.

Table 3.3 tabulates the Mean Average Precision results of various retrieval methods for all the datasets of Table 3.2 with 50 ranked retrieval trials for each dataset, capped at 1,000 results each. Bi-grams were used for the date and time data and tri-grams for the

Data Type	Naive	Naive Score	TF-IDF	Kosinov-modified	Kosinov
ISO-8601 Long hand	4.361E-5	0.475	0.241	0.475	0.475
ISO-8601 Short Date	4.717E-5	0.475	0.2183	0.4750	0.475
ISO-8601 Short Time	5.13E-5	0.209	0.128	0.209	0.209
Canadian Short Date	3.948E-5	0.0981	0.110	0.0981	0.0981
US Short Date	4.992E-5	0.0642	0.06111	0.06422	0.06422
Timing	250.62s	3,535.40s	16,918.21s	3,761.66s	N/A
Unix login name	0.052	0.396	0.374	0.367	-
Last name versus full name	0.001	0.089	0.052	0.107	0.117
Random cast in IMDB	0.0025	0.95	0.03342	0.9333	0.95
Random title in IMDB	0.0029	0.2526	0.01777	0.3469	0.2526
Random cast in MythTV	0.001	0.975	0.875	0.975	1.0
Random title in MythTV	0.002	0.8994	0.0569	0.899	0.986

Table 3.3: Mean Average Precision scores for the datasets described in Table 3.2 along with timing data.

other datasets. Half the instances in the date and time datasets were filled with random numerical values to add complexity to the problem.

The retrieval methods used are:

- Naive: Instance is retrieved if any of the q-grams matches the query; ranking is done randomly.
- Naive score: Instance is ranked according to the number of common q-grams that match the query.
- TF-IDF: Instance is retrieved and ranked according to TF-IDF scoring using an index.
- Kosinov original: Instance is retrieved and ranked according to the original Kosinov Equation 3.8, for comparison with the modified version. Since this is implemented in main memory, we do not consider computing time for comparison.
- Kosinov modified: Instance is retrieved and ranked according to the modified Kosinov distance Equation 3.9.

A question was whether the replacement of the count of unique grams within Equation 3.9 with the length of the string would significantly influence results. Furthermore, we wanted to contrast the performance of the Kosinov method with that of TF-IDF for large datasets where normalisation would be a problem.

The results of Table 3.3 show that the performance of the Kosinov method is comparable or better than the TF-IDF within this application, with about a 4.5 speedup. A close inspection of the results show that the results of the Kosinov modified and Naive score are very similar for the date and time datasets. This is because the width of the attribute and the q-gram size are constant, thus over large datasets the denominator is averaged out and Equation 3.9 simplifies to the Naive score.

These datasets were chosen as representative of the type of real world databases that we might wish to integrate. However, this comes at the cost of not completely normalising the search for instances. The collection normalisation provided by the TF-IDF method and the unique gram count provided by the original Kosinov equation are not present within our modified method.

We also experimented with the separator finding method of Algorithm 2 to change the gram generating patterns within search strings. Since it is difficult to determine the proper value of q for generating q-grams, the algorithm is an interesting way of dealing with this problem. Its use did correctly identify cases where smaller grams were necessary with the additional benefit that the lower number of individual grams increased the speed of the queries by about 20%.

Table 3.4 tabulates the performance of queries as a means of sampling the contents of a database. We use the results of the user queries presented later in Chapter 6, with a maximum retrieval limit of 1,000 tuples. Given a sample of tuples from the Myth TV database, we examined each of the 328 attributes to determine whether the values for that attribute in the sample are representative of the values for that attribute in the whole database. We performed a similar test for a sample of tuples and each of the 70 attributes in the IMDB database. For each attribute, we then calculated a χ^2 test score for each of the sample strategies and counted the number of attributes for which the confidence was above a given threshold (.90, .95, and .99, respectively).

As expected, the number of attributes that are properly sampled increases with the size of the random sample. The sampled set created using queries on both IMDB and

Sampling technique	Number of significant samples					
	MythTV (328)			IMDB (70)		
	.90	.95	.99	.90	.95	.99
10% Random	8	7	3	5	5	5
25% Random	21	13	10	16	16	14
50% Random	38	34	27	20	18	18
Query 'Harrison Ford, Blade Runner, 1982'	20	18	18	40	40	40
Query 'Joan Collins, Star Trek, 1967'	30	25	25	40	40	40
Query 'Kevin Bacon, Footloose, 1984'	36	36	36	37	37	37
Query 'Kelly Preston, Fear Factor, 2001'	19	19	14	37	37	37
Query 'Matt Battaglia, Universal Soldier II, 1998'	30	25	24	39	39	39
Query 'Adele Mara, Sands of Iwo Jima, 1949'	35	32	27	37	37	37

Table 3.4: Total number of attributes that are considered significant samples under different confidence intervals (.90, .95, .99) using a χ^2 test for each sampling approach.

MythTV databases tend to have sharp confidence intervals in that some attribute have a high significance while the rest have a low significance. This is due to the targeted nature of the queries, which tend to retrieve a number of instance from specific attributes that match the queries, while ignoring the others. Conversely, random sampling of the attribute will retrieve a number of instances from different attributes which gives us a lower confidence dispersed over all of the attributes. For our purposes of finding key / foreign key relations, separator identification and/or locating constant value attributes, both methods are non-critical.

3.5 Conclusion

In this Chapter we reviewed the problems of retrieving similar instances and sampling from relations of unknown size and distribution. Previous approaches to this problem were reviewed and their advantages and drawbacks examined. Finally, we proposed a novel means of performing information retrieval over relational database without the need for a specialised indexing mechanism. We will see in the next chapter how our approach also supports the partial discovery of potential joins with a relational databases. The performance of the retrieval method was found to be comparable to other retrieval methods

while having a significant retrieval speedup.

Chapter 4

Search for Joins

In this chapter we describe the approaches related to locating and performing joins of the relations within a single, remote database. The problem is complex: without schema information it is difficult to judge the appropriateness of joining two attributes even though their value sets completely overlap.

Consider the example shown in Figure 4.1 where a pair of relations (program and credits) may contain attributes with similar contents that can be joined. Even if we assume that no transformation is required, there exist more than eight potential single attribute join paths, such as: credits.person = people.person, credits.person = program.chan_id, credits.person = program.manual_id, credits.person = program.year, credits.chan_id = people.person, etc.

Furthermore, if both the chan_id and person attributes have similar (or identical) set values and distributions, then they may be mutually indistinguishable. Hence, determining the appropriate join based simply on the data is a difficult problem.

We first review the state of the art, examining the approaches to discovering possible joinable attributes and to verifying the correctness of the join. We follow with our observations about the performance of a straightforward method and then propose a novel method of both discovering possible joins and evaluating their appropriateness.

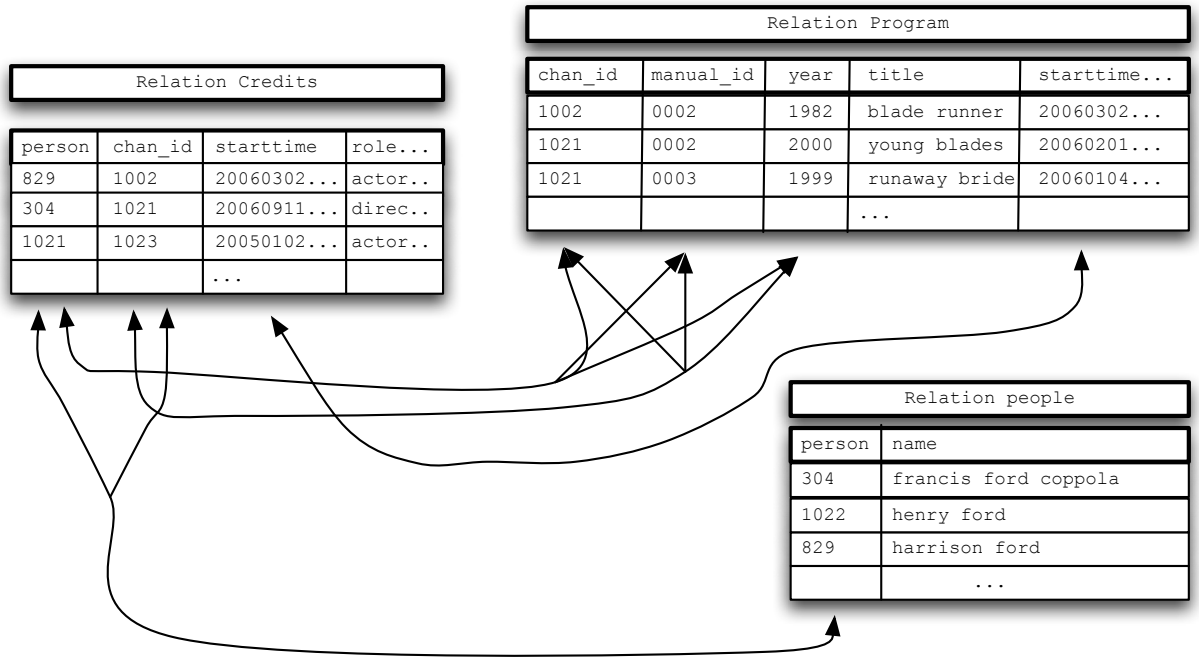

Figure 4.1: For any pair of relations, there are many possible join-able attributes.

4.1 Previous work

There are two basic means of discovering joins between relations. The first is to make use of a keyword search that identifies specific tuples within two relations and search for a pair of attributes that can join them together. The technique is to compare the set of instances contained within each of the attributes and attempt to deduce which pairs of attributes have a key / foreign-key relationship based on their distinct instance values. The earliest known work is by Ganguly et al. [42], who looked at methods of estimating the size of joins from the data and doing so for large databases with skewed distributions. Alon et al. [6] used an incremental sampling method that iteratively samples from two relations to predict the size of a joined relation. Goldman et al. [48] dealt with the off-line indexing of the number of relations that must be joined in order to link both values through a relation. Most of this work concentrated on computing all possible paths, without pruning or scoring the list of path based on likelihood.

These works, together with a paper on set similarity by Huhtala et al. [56], provided

the basis for a paper by Dasu et al. [28] on locating joins within databases. Their Bellman system primarily makes use of set theory and hash functions to compute the size of joins and the likely key and foreign key properties of attributes within database tables. They also proposed the approach of q-gram distributions and q-gram sketches to match potential attributes to one-another. While the orientation of this research was directed to the problem of finding joins, the approach mimics strongly most data-driven schema matching methods [96] in that non-exact matches between attribute instances are also considered.

The BANKS system by Aditya et al. [3] is a set of web tools used to explore disparate databases within an organisation. Using both the schema and key / foreign key constraints and a centralised off-line index, the system allows for keyword querying of multiple database and projections across databases. Resulting tuples are scored according to the keywords and according to a table normalisation factor that is computed off-line.

Agrawal et al. describe their DBXplorer [4] project, which allows the retrieval of records from databases using free-form queries. The system requires off-line indexing of the database, and the index is used to locate all exact instances of the keywords within the database. All possible join paths are created from the schema data (it is unclear whether the key / foreign key meta-data is used), so long as all query terms are present within the tuple. Results are then pruned, but they are ranked according to the join complexity required to compute them.

Hristidis et al. present their DISCOVER [55] system for keyword searching of databases. Using specialised off-line indexing for the database contents, the system ranks joins tuples based on a normalised IR metric. Joins are located according to key / foreign key database meta-data and only pursued if both the number of required joins is under a user-set threshold and all keywords would be present within the joined tuple.

Kotidis described the VIP system in use at AT&T for data integration problems in their 2006 paper [66]. The system takes a holistic approach to the location and use of joins within a database and among databases. While all joined paths are taken into account by the system, these are independently scored by a number of data quality metrics. For example, a join path may be evaluated by the fraction of tuples generated that violate other database's constraints or the semantic distance of the schema attribute names.

Kimelfeld and Sagiv provided a framework for the theoretical study of keyword searches from graphs within databases [61, 62]. Their ranking of possible join paths is based on the

distance between pairs of keywords.

Liu et al. [74] also studies the retrieval of tuples from databases using join searches, with a variety of different retrieval scoring and normalisation schemes. Whole keywords are used to search both the schema and the data, and the results are ranked using an IR-style score, with possible joins located using schema key and foreign key data. The paper provides several performance comparisons between various normalisation methods. Unfortunately, the authors' experimental design did not take into account the validity of the tuples retrieved and assigned a ground truth based on the presence of the query terms within the retrieved tuples only.

Most recently, Mayssam et al. [101] produced the KITE system, which can be used to retrieve tuples from multiple relational databases using keywords. Off-line processing is used to index terms within all databases and to provide normalisation to the searches. Its approach to pruning the number of possible joins between relations is one based on a cost model which is assumed to be provided by the database optimiser. However, a user defined benefit function is also assumed to be provided to offset the costs of different join strategies. Furthermore, only networks of joined tuples that contain all of the search terms may be returned as answers, further pruning the results.

Because the method does not depend on the knowledge of keys and foreign keys, the authors attempted to add a validation of the joined tuples by using a schema matching technique called Simflood [78]. This improved the precision of the returned results from 26-64% to 80-96%. Several different strategies were proposed for the generation of the results, each enforcing a different tradeoff between the complexity of the joins and the number of tuples being returned.

A concern is that all of these approaches return a scoring or ranking result that can compare potential joins, but gives no indication of confidence on the appropriateness of the join itself. In most of these systems, the generation of reasonable joins is based on the use of threshold factors, such as: the minimum number of joined tuples, the maximum number of joined tuples and the maximum number of joins performed. Some of the systems described, such as KITE [101] and DBXPLORER [4], are based on mitigating the computational load of performing the join, yet none of these constraints address the problem of the validity of performing the join.

Name	Uses schema schema	Uses keys	Join scoring	Join pruning	Keyword search
BANKS	Yes	Yes	IR	No	Some Keyword
DISCOVER	No	Yes	IR	Threshold	All keywords
VIP	Yes	Yes	Various	Various	Keyword
KITE	Yes	No	User / database complexity	Simflood Simflood	Word lookup Word lookup
Liu et al.	Yes	Yes	IR	None	Keyword, phrase
DBXPLORER	Yes	?	Complexity	No	All keywords
Kimelfeld et al.	N/A	N/A	Distance	N/A	N/A

Table 4.1: Characteristics of previously known systems used to locate joins within databases.

4.2 Research Problems

The problem of finding joins differs from matching in that matching is usually performed across databases while joins are performed within databases. Intuitively, we know that joins within the databases should have a higher standard of quality because the identifiers that form the key and foreign-key relationships will match exactly.

In our work, we also use a free-form query to allow the user to specify information that lies within his integration interest. We have already reviewed in Chapter 3 the elements that are required to query a database for free-form queries. Here we review how the results from a free form query can be joined into a single relation.

Most of the current methods concern themselves with predicting which attribute is the key or foreign key within the join and the number of tuples possibly generated. The actual detection of joins is done by calculating the overlap between the two sets of attribute instances. However, the metrics used to assign validity to a certain join is done by simply assigning a minimum and maximum number of tuples generated through the join. The unaddressed question is how to assign thresholds when we do not know what is appropriate for this database.

Our solution is to search potential joins for pairs of attributes known to be linked through a relation. Hence, not only do we locate joins within the database, but we now ensure that the tuples created match those that are already present within another database. That is, we rely on relation information from another database to search for the join rather

than merely examining the overlap between the two column values.

The second problem is how to locate potential joins when representative sampling of the relations within the database is difficult to achieve. We circumvent this problem by using all of the retrieved instances from each relation that had already been searched to form a sample skewed to the query terms. The comparison of these skewed samples is then used to locate joins.

We note that for illustrative purposes only we present schema information within diagrams and tables, despite this information not being used at the algorithmic level for decision making. Also, the reader will notice many redundant operations that would be combined within an implementation, as well as unmentioned caching opportunities. We refrain from doing so here for clarity.

4.3 Basic methodology

As mentioned in Chapter 1, we require the user to enter a trial query that represents information of interest to the user. From a database integration perspective, this query allows the integration to be localised within the foreign database D^{foreign}, without attempting to integrate relations that are judged irrelevant.

An example query, $\mathcal{Q}$, could be "Blade Runner Harrison Ford Actor" that references a particular actor in a movie. For example, this query would focus our integration efforts on the movie title and credits section of a database, and perhaps leave out most of the movie review discussion and reviewer user information relations.

We use this simple keyword term query to locate the specific attributes of interest within both the foreign database (D^{foreign}) and the local database (D^{local}).

We start by querying the table R^{local}, from our local database, for the user's query terms $\mathcal{Q}$. Because this is a known, local database of our own design, we also make the assumption that we can match attributes to query terms and that a tuple can be retrieved that matches most, if not all, of the query terms. Through this method, we also resolve any ambiguous n-to-1 relationships, so that the two query terms "Harrison Ford" are known to refer to the same attribute. To ensure that the terms are valid, query terms that are not found within the local database are simply ignored.

Data: Relation R^{local} from database D^{local} and user query set $\mathcal{Q}$.

$\mathcal{MAP} \leftarrow 0$ /* Initialise query term to attribute map. */

$\mathcal{T}^{\text{local-ex}} \leftarrow 0$ /* Initialise example tuples holder. */

foreach Q_i in $\mathcal{Q}$ **do**

  Find A_x^{local} that matches Q_i in R^{local};

  if *!found(Q_i)* **then**

    /* Remove query terms that we cannot find in the local database. */

    Remove Q_i from $\mathcal{Q}$;

  else

    /* Store mapping from this query term to one or more local attributes. */

    $\mathcal{MAP} \leftarrow \text{map}(Q_i \rightarrow A_x^{\text{local}})$

  end

end

/* Retrieve the local tuples for the user query. */

$T_1^{\text{local-ex}} \leftarrow \pi_{\forall \mathcal{A}^{\text{local}} \in \mathcal{MAP}} \left(\sigma_{\forall \mathcal{A}^{\text{local}} \mathcal{MAP} \mathcal{Q}} \left(R^{\text{local}}\right)\right)$;

for $x = 2$ to M **do**

  $T_x^{\text{local-ex}} \leftarrow \pi_{\forall \mathcal{A}^{\text{local}} \in \mathcal{MAP}} \left(\sigma_{\forall \mathcal{T}^{\text{local}} = \mathcal{T}_{\text{rnd}}^{\text{local}}} \left(R^{\text{local}}\right)\right)$;

end

return *Query term to attribute mapping $\mathcal{MAP}$, sampled set of tuples from local database $T^{\text{local-ex}}$ with first tuple matching user query $\mathcal{Q}$;*

Algorithm 3: Find mapping of user query to local database and retrieve additional sample of tuples to validate any potential joins.

This process is depicted in Algorithm 3, which returns the attributes matching each query term and select other tuples ($T^{\text{local-ex}}$) to verify the validity of the joins being considered. The number of example tuples retrieved and the sampling method are not critical; we select 10 tuples at equal intervals. Another method that we specifically did not pursue, was the re-querying of the foreign database D^{foreign} using the sampled values. While additional searches with different instances would be useful to validate the networks, this would come at the cost of searching the database another $j * i$ times more than the cost of simply validating potential joins through a localised query of the join with the sampled

values.

Data: A user query $\mathcal{Q}$, foreign database relations $\mathcal{R}^{\text{foreign}}$.

$\mathcal{I}_i^{\text{query}} \leftarrow 0$;
$\mathcal{I}_{jl}^{\text{attrib}} \leftarrow 0$;
for $j = 1$ to J **do**
    /* Process all of the database at least once. */
    for $l = 1$ to L_j^{foreign} **do**
        for $i = 1$ to I **do**
            /* Get entire tuple with top-k query of attribute. */
            $\mathcal{T}_{jli}^{\text{foreign\_match}} \leftarrow$ query top-k $R_j^{\text{foreign}} \pi A_{jl}^{\text{foreign}}$ top-k Q_i;
            if *count(substr(*$\mathcal{T}_{jli}^{\text{foreign\_match}}$*,*$Q_i$*))* $= k$ **then**
                large$(Q_i) \leftarrow true$;
            end
            $\mathcal{I}_i^{\text{query}} \leftarrow \mathcal{I}_i^{\text{query}} \cap \mathcal{I}_{jli*l}^{\text{foreign\_match}}$;
            /* Assign retrieved data to query term. */
            for $x = 1$ to L_j^{foreign} **do**
                /* Opportunistically, retrieve instances for matching. */
                $\mathcal{I}_{jx}^{\text{attrib}} \leftarrow \mathcal{I}_{jx}^{\text{attrib}} \cap I_{jli*x}^{\text{foreign\_match}}$;
            end
        end
    end
end
return *Set* $\mathcal{I}_i^{\text{query}}$ *of all instances matching query* Q_i *and set* $\mathcal{I}_{jl}^{\text{attrib}}$ *of all known instances of attribute* A_{jl}^{foreign}.

Algorithm 4: Retrieve instances from all of the attributes and relations for each of the query terms.

We then query every attribute from every relation within the foreign database D^{foreign} for each term Q_i within the query $\mathcal{Q}$, as shown in Algorithm 4. We make use of the retrieval method as described in Chapter 3.3 to retrieve the top-k most similar instances. We initially set k to an arbitrary 1,000 tuples, but the choice of this value is reviewed later in Chapter 6. We do add one caveat: should the query term retrieve k instances that are

exactly the same or substrings of the query term, we mark it as *Large*. This indicates that the query term is too common to use in a join network immediately, we must wait until we can effectively re-query the database with another partial network for the query to be effectively used.

For I query terms, top-k retrieval gives us a maximum number of $k * i$ scored instances that we rank to find the instances most likely to be of interest to the user. But since we must scan the entire database at least once to find those terms, we also use the query results to build a skewed sample, or directed model, of each individual relation attribute within the foreign database.

As mentioned in Chapter 3, this provides an elegant solution to the problem of sampling relations and attributes. Whereas sampling earlier would have been done as a separate step in which relations would have been sampled, we piggy-back onto the retrieval operation to obtain a directed, biased sample using the user query itself. It is also a sampling method that does not require any knowledge of the number of tuples within the relation, which is an operation that can be expensive in certain circumstances.

Data: Set $\mathcal{I}_{jl}^{\text{attrib}}$ of all known instances of attributes A_{jl}^{foreign}.
$\mathcal{SIM}() \rightarrow 0$ **foreach** pair { *(j1,l1), (j2, l2)* } **do**
  $a, b \leftarrow 0$;
  $a \leftarrow \mathcal{I}_{j1,l1}^{\text{attrib}} \text{overlap} \mathcal{I}_{j2,l2}^{\text{attrib}}$;
  $b \leftarrow \mathcal{I}_{j1,l1}^{\text{attrib}}$ cosine $\mathcal{I}_{j2,l2}^{\text{attrib}}$;
  $\text{sim}(\mathcal{I}_{j1,l1}^{\text{attrib}}, \mathcal{I}_{j2,l2}^{\text{attrib}}) = \frac{a+b}{2}$;
end
Sort $\mathcal{SIM}()$ on score;
return *Ranked set* $\mathcal{SIM}()$ *of similarity between all attributes in* D^{foreign}.

Algorithm 5: Compare the sampled distribution for each relation pair and rank them from most similar to least similar.

Although, the sample is not a representative sample, we do obtain k instances for i query terms for each attribute. This is sufficient for us to use attribute matching and join search algorithms using this $k * i$ set of instances.

Intuitively, we know that some attributes will have fewer than the maximum of $k * i$ instances, as some query terms will be unmatchable to the attribute contents (e.g.: 'arnold'

can never match a numeric attribute). These smaller, or empty, sets are still useful: their distribution will be substantially different to that of large sets and prevent their match.

As reviewed in Algorithm 5, these set values are compared for each attribute pair and a score is computed. By ranking the score for each pair of attributes, we can obtain an order with which the potential join relations should be searched. The actual method to compare the samples is not critical, and we use an aggregate of a cosine similarity test and value overlap similarity test to assign similarity. Furthermore, attributes that have less than three distinct values are placed at the tail of the similarity queue. This is done to prevent these columns from matching every possible join conditions.

We then rank $\mathcal{I}_i^{\text{query}}$, the set of the instances that have been retrieved for each search term Q_i, based on their normalised Kosinov score (see Chapter 3.3), as in Algorithm 6. This is done so that the tuples that are likeliest to be relevant to a query term are matched against each other for a potential join first.

Data: Set $\mathcal{I}_i^{\text{query}}$ of retrieved instances for query $\mathcal{Q}$.
$\mathcal{NET}() \leftarrow 0$;
foreach I in $\mathcal{I}$ **do**
    Sort I_i^{query} according to Kosinov score;
    NET(Q_i) $\leftarrow$ distinctR^{foreign},A^{foreign},I_i^{query};
end
Sort $\mathcal{NET}()$ according to top-k Kosinov score.;
return *Network* $\mathcal{NET}()$*, sorted according to top scoring instances.*

Algorithm 6: Find highest scoring instances for each query and rank all queries according to their highest instance score.

The specific relation and attribute from which these instances were retrieved is termed a "triplet" and together with the ranking of the specific instance allows us to locate a distinct area of the database to search. In Table 4.2 for example, we list possible matches for the query terms "harrison" and "blade", query terms Q_1 and Q_2 respectively.

Because the triplets are ranked according to their similarity to the query term, we can use this as an ordering with which to search different areas of the databases for the correct attribute and instance. The first query term "harrison" is meant to refer to the actor "Harrison Ford", but also matches another actor "Harrison Paul" as well as a movie title

#	Q_1	Rel.	Attr.	Instance	Score
1	Harrison	People	Person	Harrison Paul	1.02
2	Harrison	People	Person	Harrison Ford	1.02
3	Harrison	Program	Title	Meet the Harrisons	0.88
4	Harrison	People	Person	Harris Kevin Lyle	0.48
...	...	...	...	...	...

#	Q_2	Rel.	Attr.	Instance	Score
1	Blade	Program	Title	Blade Runner	1.02
2	Blade	Program	Title	Blades	1.02
3	Blade	People	Title	John Blade	0.98
4	Blade	Reviews	Text	...the blade then...	0.48
...	...	...	...	...	...

Table 4.2: Example results from a retrieval.

"Meet the Harrisons". Therefore we would first search for a join under the assumption that the proper mapping for Q_1 would be the relation People, the attribute Person and instance "Harrison Paul". Failing this, we would then try instance "Harrison Ford" and then relation Program, attribute title, instance "Meet the Harrisons" and so forth. The results of all possible joins of Table 4.2 would be similar to those listed in Table 4.3.

Q_2			Q_1			Join
Rel.	Attr.	Instance	Rel.	Attr.	Instance	possible?
Program	Title	Blade Runner	People	Person	Harrison Paul	$\neq$
Program	Title	Blade Runner	People	Person	Harrison Ford	$=$
Program	Title	Blade Runner	People	Title	John Blade	$\neq$
Program	Title	Blade Runner	Reviews	Text	...the blade then...	$\neq$
...	...	...	...	...	...	...
Reviews	Text	...the blade then	Program	Title	Meet the Harrisons	$\neq$
...	...	...	...	...	...	...

Table 4.3: Different possible joins of the selections in Table 4.2 being attempted.

The scores of the instances are also used to provide an order for the query terms to be used within the search. This is done so that when we begin searching for joins, we do so using the queries for which instances are the most similar to the original query term Q_i, in a manner similar to the work of Brin [16] in finding connections between sets of query terms in web-pages. We call one or more query terms and their associated triplet, which can be linked by a join, a "network". Figure 4.2 shows an example of networks for the query terms "harrison", "blade" and "actor". Note that it is the specification of specific triplets that constrains the appropriateness of a join. The relations binding the triplets ensures that only joins that can assign the correct values to the instances are kept. The approach of having a network reference an attribute, tuple or specific instance is not unlike the approach of Srivastava and Velegrakis [105], however whereas they make use of the complete meta-data we only make use of the relation.

As an example, a join between the query term "blade" (Network 4) and the query term "Ford" (Network 1) is identified, because both of the tuples they point have an attribute whose value is common. In general, a join is deemed to be appropriate when

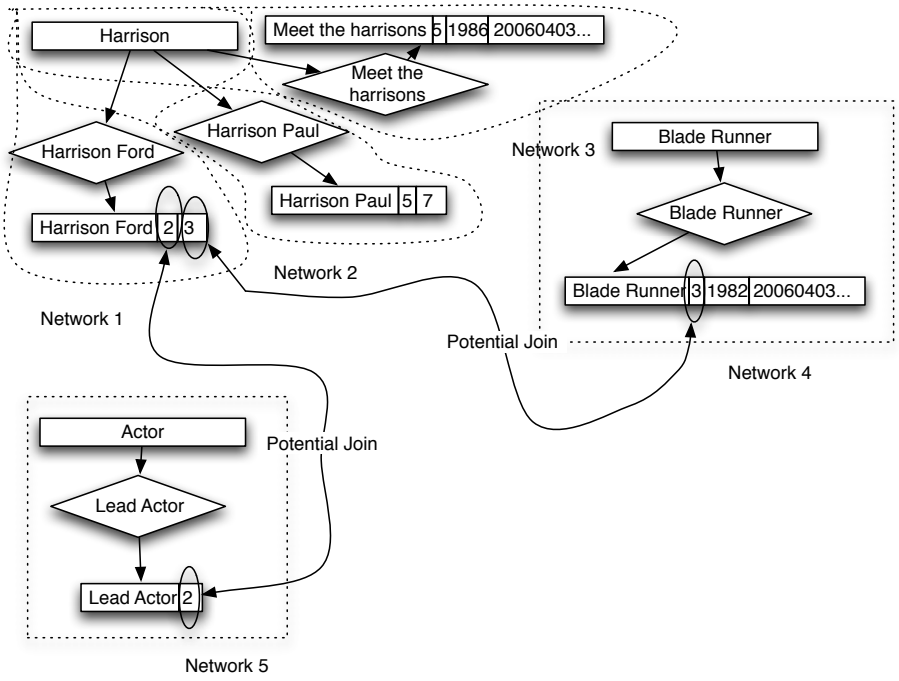

Figure 4.2: Several different join possibilities exist for each possible assignment of Q_i.

a shared attribute value makes it possible to relate both specific tuples through the join. Conversely, it is not appropriate to directly join query term "blade" (Network 4) with query term "Actor" (Network 5), as none of the attribute values for either networks allow them to form a join.

We use the word "appropriate" and not the word "correct". The join may appear correct because it links both query instances properly, but the specific attribute providing the join may do so serendipitously. We will therefore need to further verify these joins at a later time to increase the likelihood that they are meaningful.

We begin the search for joins that are appropriate with the highest ranked query term according to $\mathcal{I}_i^{\text{query}}$. The query term ranking does not guarantee that the first query-triplet mapping will be correct, but it chooses the query with the likelihood of obtaining the correct triplet the soonest.

Data: Set $\mathcal{NET}()$ of all possible networks, where NET($Q_{(...)}$) contains all of the potential networks linking one or more queries $Q_{(...)}$.

```
1  NEWNET() ← 0, LARGENET() ← 0;
2  for a = 1 to size(NET()) do
3      if Large(NET()_a) then
4          LARGENET() ← NET()_a;
5      end
6      NEWNET() ← NET()_a;
7  end
8  done = false; NET() ← NEWNET();
9  while done do
10     done = true;
11     for a = 1 to size(NET()) do
12         for b = 2 to size(NET()) do
13             for x = 1 to size(NET()_a), y = 1 to size(NET()_b) do
14                 if (A_a^foreign != A_b^foreign) then
15                     Join = NET()_ax ⋈ NET()_by;
                       /* Join must exist, match at one tuple of T^local−ex and
                          not expand the number of tuples.                    */
16                     if (Score(Join) > 0) ∧ (count(Join) > 0) ∧ (count(Join) <
                       max(count(NET()_ax),count(NET()_by)) then
17                         NEWNET() ← Join;
18                         done = false;
19                     end
20                 end
21             end
22         end
23     end
24 end
25 NET() ← NEWNET();
26 return Sets NET() and LARGENET() of all possible networks.
```

Algorithm 7: Initial search for simple query to query joins. Note that the *Score* function is detailed in Algorithm 11 at the end of this chapter.

We direct the search as a concurrent search for instances that are similar to the query terms and for pairs of attributes that can function as joins. The intuition behind this strategy is that most database joins are simple key / foreign-key relationships. Join paths requiring one or more intermediate relations are to be expected, however it is likely that by finding the simple joins first our constraints will be more effective at locating complex joins.

To locate simple joins, we enumerate every pair of triplets that can join any two query terms. Since the tuples were entirely retrieved along with the similar instances in Algorithm 5, we can perform this operation within the integrator's main memory. This process is described in Algorithm 7, where the triplets of each possible pair of query terms is checked for a possible joining attribute.

Each of these operations involves finding common instance values between both queries to enumerate all known appropriate joins. The resulting set of networks described in Algorithm 7 would contain networks similar to $\text{NET}(\text{harrison}, \text{blade}) = A^{\text{foreign}}_{j=\text{people},l=\text{movie\_id}} \bowtie_{?=?} A^{\text{foreign}}_{j=\text{program},l=\text{movie\_id}}$ for Q_1 and Q_2 and resembles the process depicted in Figure 4.3.

To enforce a measure of correctness on the proposed join, we make sure that the instance values within each potential join can only be one of those previously retrieved. Hence, as in Figure 4.3, we search for a pair of join attributes that allow the attribute instances of Table 4.2 to be joined across both relations into one of the appropriate answers of Table 4.3. Searching for the network would entail a query similar to $A^{\text{foreign}}_{j=\text{people},l=\text{movie\_id}} \bowtie_{?=?} A^{\text{foreign}}_{j=\text{program},l=\text{movie\_id}}$ where $A^{\text{foreign}}_{j=\text{people},l=\text{person}} =$ 'Harrison Paul' $\vee$ 'Harrison Ford' $\vee$ 'Meet the Harrisons' $\vee$ 'HarrisKevin Lyle' $\wedge$ $A^{\text{foreign}}_{j=\text{program},l=\text{title}} =$ 'blade runner' $\vee$ 'blades'.

Therefore, we would look for a pair of join attributes from both networks. In this case, the top two potential instances of person would be "Harrison Ford" or "Harrison Paul" and the instance of title within the second relation must be "Blade Runner". This gives us a solution that identifies the appropriate join path, as depicted in Figure 4.3.

Algorithmically, we need several other conditions to deem a potential join as being appropriate, as several spurious potential joins will likely be present. We add two conditions for a potential join to be considered appropriate: the join may not create more tuples than the largest parent network and must include at least one additional tuple from $\mathcal{T}^{\text{local}-\text{ex}}$, according to Algorithm 11.

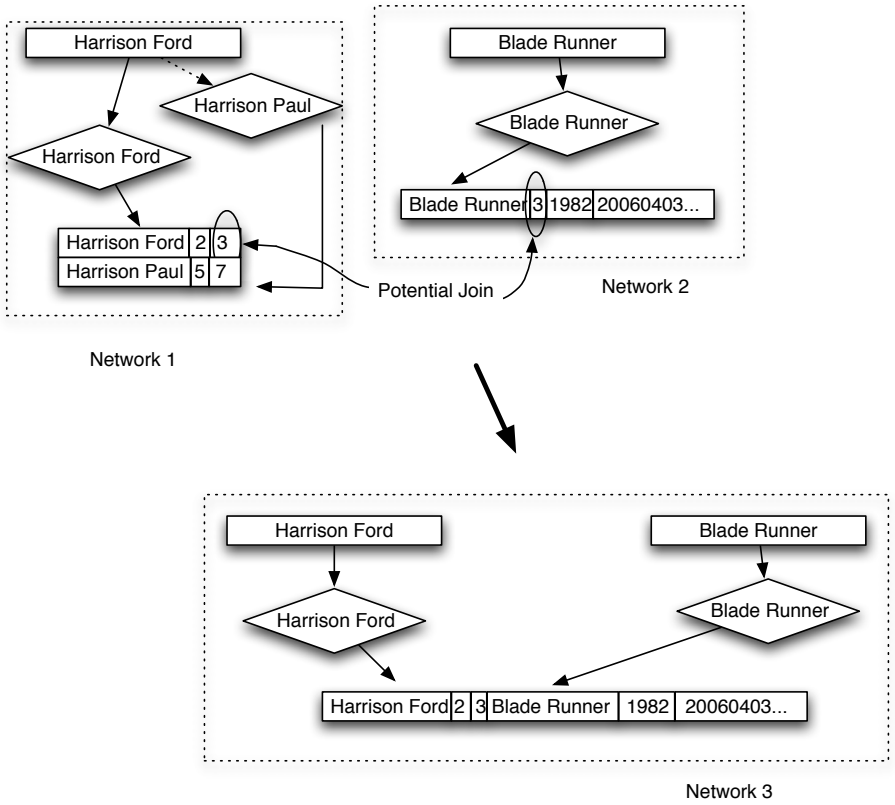

Figure 4.3: Search for a possible join.

The intuition from this requirement is that the new network should not generate more ambiguity. Hence, if the proposed join of both networks generates more tuples than either network did, then the constraints have been relaxed instead of restricted and the join is inherently wrong. The sampled tuples provide a measure of testing whether the join is random or likely to be appropriate; if the join can support one local support, it should also support at least another one. We can score each proposed network by counting the number of sampled tuples that can be matched through the join. However, we do not do so at this point since the initial joins created are too simple for the scores to mean anything.

When a pair of networks has been selected as a joined pair, we continue the search for additional join attributes by using the remaining unattached networks, as in Figure 4.4. Since the previous query provided us with several sampled relations that match the previously selected joined attribute, we can directly query the next candidate relation looking

for the correct joinable attribute. This prevents us from having to perform a complex join across relations that were previously queried. The process continues until we have joined all of the networks or until we have exhausted the combinations of networks that can be joined.

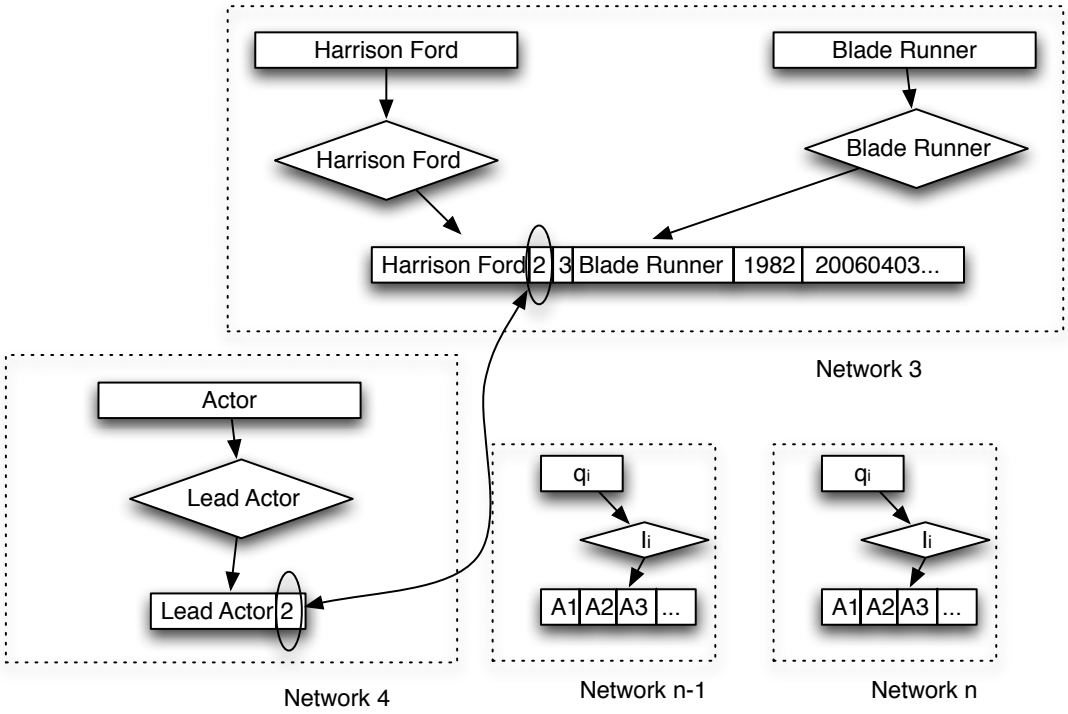

Figure 4.4: We iterate through the rest of the joinable attributes looking for additional networks to join.

We intentionally set the large networks aside in Algorithm 7. This is done because the large networks are unlikely to capture all of the appropriate instances that are similar to a query term. Therefore in a second process, Algorithm 8, we re-visit these networks but do so with the relational database. Since the search is done directly on the database management system, we can now afford to make use of these networks while relying on the selectivity of the smaller networks to test for appropriate joins.

Data: Set $\mathcal{NET}()$ and $\mathcal{LARGENET}()$ of all possible networks, where $\text{NET}(Q_{(...)})$ contains all of the potential networks linking one or more queries $Q_{(...)}$.

$\mathcal{NEWNET}() \leftarrow 0$;
for *a = 1* to *size($\mathcal{LARGENET}()$)* **do**
  for *b = 1* to *size($\mathcal{NET}()$)* **do**
    for *x = 1* to *size($\text{LARGENET}()_a$)* **do**
      for *y = 1* to *size($\text{NET}()_b$)* **do**
        if *($A_a^{\text{foreign}} != A_b^{\text{foreign}}$)* **then**
          Join = $\text{LARGENET}()_{ax} \bowtie_{?=?} \text{NET}()_{by}$;
          `/* Join must exist, match at one tuple of` $\mathcal{T}^{\text{local-ex}}$ `and not expand the number of tuples. */`
          if *(Score(Join) > 0) ∧ (count(Join) > 0) ∧ (count(Join) < max(count($\text{LARGENET}()_{ax}$),count($\text{NET}()_{by}$))* **then**
            $\mathcal{NEWNET}() \leftarrow$ *Join*;
          end
        end
      end
    end
    $\mathcal{NEWNET}() \leftarrow \text{NET}()_b$;
  end
  $\mathcal{NEWNET}() \leftarrow \text{LARGENET}()_a$;
end
$\mathcal{NET}() \leftarrow \mathcal{NEWNET}()$;
return *Sets $\mathcal{NET}()$, $\mathcal{NEWNET}()$ of all possible networks.*

Algorithm 8: Search for additional networks that do not discriminate well within local results.

Since much of the low-hanging fruit has already been combined into networks by the integrator, we can mitigate the potential cost of querying the database itself with the constraints imposed by known networks. Hence, while querying a large network might be expensive in the number of tuples involved, only a fraction of these will meet the requirement of a join with another network with specific instance values of specific attributes.

The constraints imposed by the smaller networks also provide information to the database query optimiser as to the optimal way to search for the join.

Furthermore, we note that we can often process multiple proposed joins within the same query, as presented in Query 4.1. This allows for the testing of multiple hypotheses while reducing the number of times the relations are traversed by the database.

Query 4.1 We can make direct use of the database itself to search for multiple join possibilities concurrently.

```
Select R1.person, R1.role_id, R1.movie_id, R2.title, R2.date,
 R2.movie_id, R2.year from R1,R2 where
(R1.movie_id=R2.movie_id OR R1.movie_id=R2.date OR
R1.role_id=R2.year ...) AND ( R1.person='Harrison Ford' OR
R1.person='Harrison Paul'...) AND (R2.title='Blade Runner'
OR R2.title='Blades')
```

By inspecting the results of the query we are able to determine which of the possible join pairs actually functioned as joining attributes. The ability to search for multiple hypothetical joins also allows us to locate multi-attribute keys where multiple attributes are needed to form the join relations. However, this has the tendency to generate join conditions which include key / foreign key conditions that are not useful, such as those containing constant value attributes. Therefore, we make use of Algorithm 9 that verifies the validity of all of the conditions within the join.

Data: A network NET() who's join conditions are to be pruned.
foreach $\bowtie \in$ NET() **do**
  foreach $Path \in \bowtie_{\text{Path}_1,\ldots,\text{Path}_n}$ **do**
    if $G_{\text{count}(*)}\ (\ \bowtie\ ) = G_{\text{count}(*)}\ (\ \bowtie - Path\ )$ **then**
      $Join = Join$ - $Path$;
    end
  end
end
return *Network NET() free of unnecessary join conditions.*

Algorithm 9: Iteratively make sure all of the key / foreign key relationships are needed and not tautaulogies on constant value attributes.

Up to now, all of the joins being considered were assumed to be network to network joins, without intermediate relations linking them. The condition we use to determine whether to continue or terminate the search is whether we have obtained a network that joins all of the active query terms into a projection. If one or more of these networks are available, we terminate the process and proceed to translate the relation.

Algorithm 10 shows both this decision condition and the search for networks of increased complexity. We consider all possible joins at a distance of 1-relation, then 2-relations and so forth. The algorithm is very similar to Algorithm 8 where we attempt to merge networks until we exhaust the search space. However in this case, we compute all of the paths between the attributes of both networks through one or more intermediate relations.

We compute the ordering of the path generation using the similarity scores computed previously in Algorithm 5, grouping attribute-to-attribute paths by relations so that we may use the same aggregate search described for Query 4.1. We compute all possible networks for a given join distance and terminate only when a complete network has been found.

The final result is a query of the foreign database that logically joins all of the relations necessary to the integration process, as in Figure 4.5. This large virtual relation then remains to be translated to the relation R_{local} within our local database.

4.4 Experimental Results

In this section, we review the performance of two schema matching methods used to predict key / foreign key relationships within a database. Experiments making use of the network finding method proposed will be reviewed in Chapter 6, at the end of the thesis.

Table 4.4 shows the performance of both retrieval and sampling methods as a means of providing a sampled set that can be used to locate joins. We use a set of randomly perturbed titles from the MythTV database as with the previous experiment. The Mean Averaged Precision (MAP) score is calculated from the ranked list of possible joins returned by the method, scored against the list of key / foreign key relationships within the MythTV database. The two methods here are the set ressemblence method (value overlap) reviewed by Dasu et al. [28] and cosine similarity between the sets of attribute instances.

Overall, the results of Table 4.4 represent an interesting decision to be taken by the

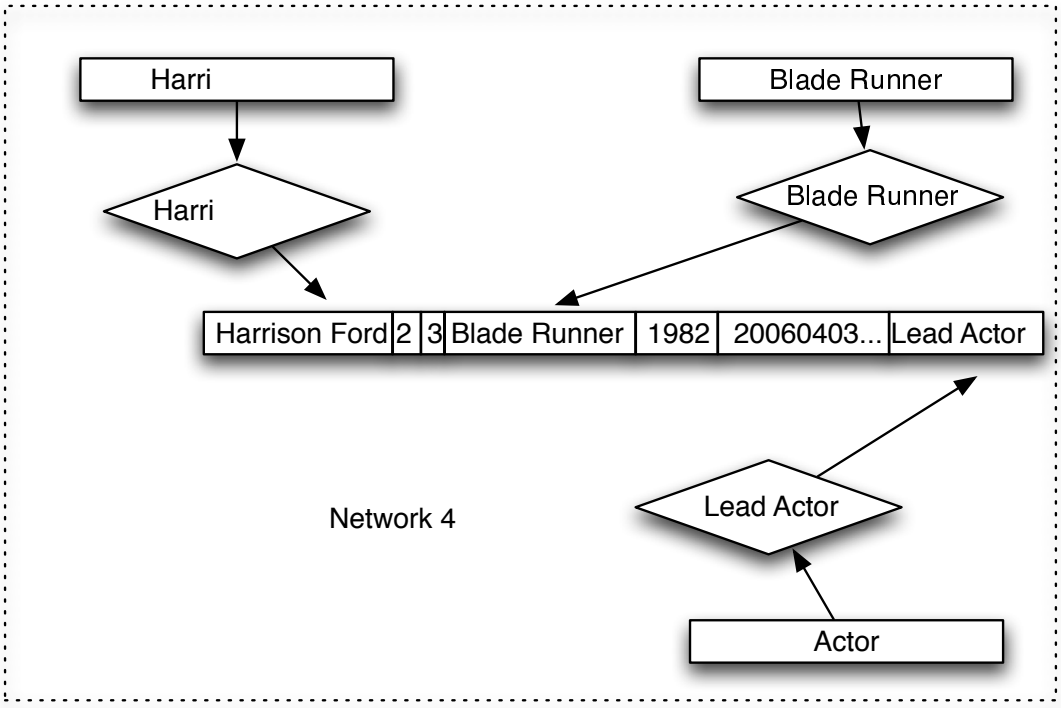

Figure 4.5: The final result of the join search is a solution that links the different queries into a unique table.

Retrieval Method	MAP (Overlap)	MAP (Cosine)
Kosinov modified	0.003167	0.003720217
10% sampling	0.000324	0.0005634
all data	0.000775054	0.000505279

Table 4.4: Mean Average Precision scores for possible key / foreign key rankings with the MythTV database.

designer. The overlap method tends to identify 'perfect' key / foreign key relationships within the database. These tend to favour numerical identifiers with a restricted set of values, and while the results of Table 4.4 favour it, it tends to return a large number of false positive. On the other hand, the cosine method of locating key / foreign key relationship will favour attribute pairs that have a good contextual or similarity linkage that is evident in the returned results.

For this reason the overlap method may be a better choice for attribute mapping applications instead of key / foreign key relationship searches. In either case, the performance of the join predictors is very poor. Both of these methods are meant for schema matching and we make use of them to prioritise our search with a likelihood indicator.

These results are consistent with some of the results by Bohannon et al. [14]. They reported that attribute matching algorithms could differentiate between related attributes such as sale_price and total_price because of patterns in the prices, such $5.99, $2.99 or $6.00 and $10.00.

In our case, we obtained similar results in locating contextual similarities between attributes, such as human readable descriptions or timestamps, but this was a disadvantage. Since we were attempting to locate key / foreign key relationships within the same databases, and which are all numerical in nature, these semantic relations served only to clutter our search space.

4.5 Conclusion

In this chapter we reviewed algorithms that search for joins within a foreign database. Our approach does not require a specialised index of the database contents, nor schema information, and minimises the amount of information retrieved from the foreign database. Previous work concentrated on reducing the number of possible joins and time of computation; from our survey of previous work, we believe that the enforcement of a correctness concept during the search for joins is novel work.

With a global relation extracted from the foreign database, we now turn our attention to the problem of translating the relation to the same structure that is in use in our local database.

Data: Set $\mathcal{NET}()$ from Algorithm 8 and ranked similarity scores $\mathcal{SIM}()$ for all attribute pairs.

```
1  distance ← 1;
2  done = false;
3  while (done) ∧ (!∀active(Q) ∈ NET()) do
4      done = true;
5      for a = 1 to size(NET()) do
6          for b = 2 to size(NET()) do
7              for x = 1 to size(NET()_a), y = 1 to size(NET()_b) do
8                  foreach Path in = SIM(x, y) of length distance do
9                      if (A_a^foreign ! = A_b^foreign) then
10                         Join = NET()_ax ⋈ Path ⋈ NET()_by;
11                         if (Score(Join) > 0) ∧ (count(Join) > 0) ∧ (count(Join) <
                           max(count(NET()_ax),count(NET()_by)) then
12                             NEWNET() ← Join;
13                             done = false;
14                         end
15                     end
16                 end
17             end
18         end
19     end
20     NET() ← NEWNET();
21     if !∀active(Q) ∈ NET() then
22         done = false;
23         distance + +;
24     end
25 end
26 return Set NET() of currently known networks.
```

Algorithm 10: Attempt to find networks that have intermediate relations within the joins.

Data: $\mathcal{T}^{\text{local}-\text{ex}}$ a set of additional tuples from R_{local} mapped to $\mathcal{Q}$, NET() a network that joins several attributes of $R_{foreign}$ through $\mathcal{Q}$.

for m=2 to M **do**
  $c = G_{\text{count}(*)}(\sigma_{\forall \text{Q} \in (\text{NET} \wedge \mathcal{MAP})} (\sigma_{\text{Q} \longmapsto A^{\text{foreign}}} (\sigma_{A^{\text{local}} \longmapsto \text{Q}})))$;
  if $c > 1$ **then**
    $s++$;
  end
end
return s *the number of example queries that support this network.*

Algorithm 11: Score(Net) Score the potential network based on the number of additional sampled relations that it can successfully retrieve.

Chapter 5

Schema translation methods

Locating joins is about discovering the structure of the database and projecting it into a single relation. This relation must then be integrated with the relation within the local database. There are two basic elements to the process. Matching is the first, where attributes from each relation are linked together. The second deals with the translation, or transformation, of the data from one representation to another between each set of attributes.

Figure 5.1 is an example of the type of problem that we wish to solve. The composite relation, which we term R^{join}, from the foreign database D^{foreign} is similar to the relation of the local database R^{local}, but not all of the same attributes are present: R^{local} contains new information, such as instance "2:00", that is not present within R^{foreign}. Furthermore, not all of the data representations are the same: $A^{\text{local}}_{\text{role}}$ may refer simply to an 'actor' within a movie, while $R^{\text{foreign}}_{\text{role}}$ makes a 'lead actor' / 'actor' distinction that is unavailable within R^{local}.

Similarly, multiple standards exist to represent the same information in a concise format, and understanding which representation is in use takes time. For example, the Open Group lists 22 locales, each with its own typeset standard for date and time information [87]. In this chapter, we focus on the location of the matching attributes and their proper translation from one representation to another.

This is why we are investigating methods that can automate the search for matching information within a database schema and infer a mechanism for the translation of the

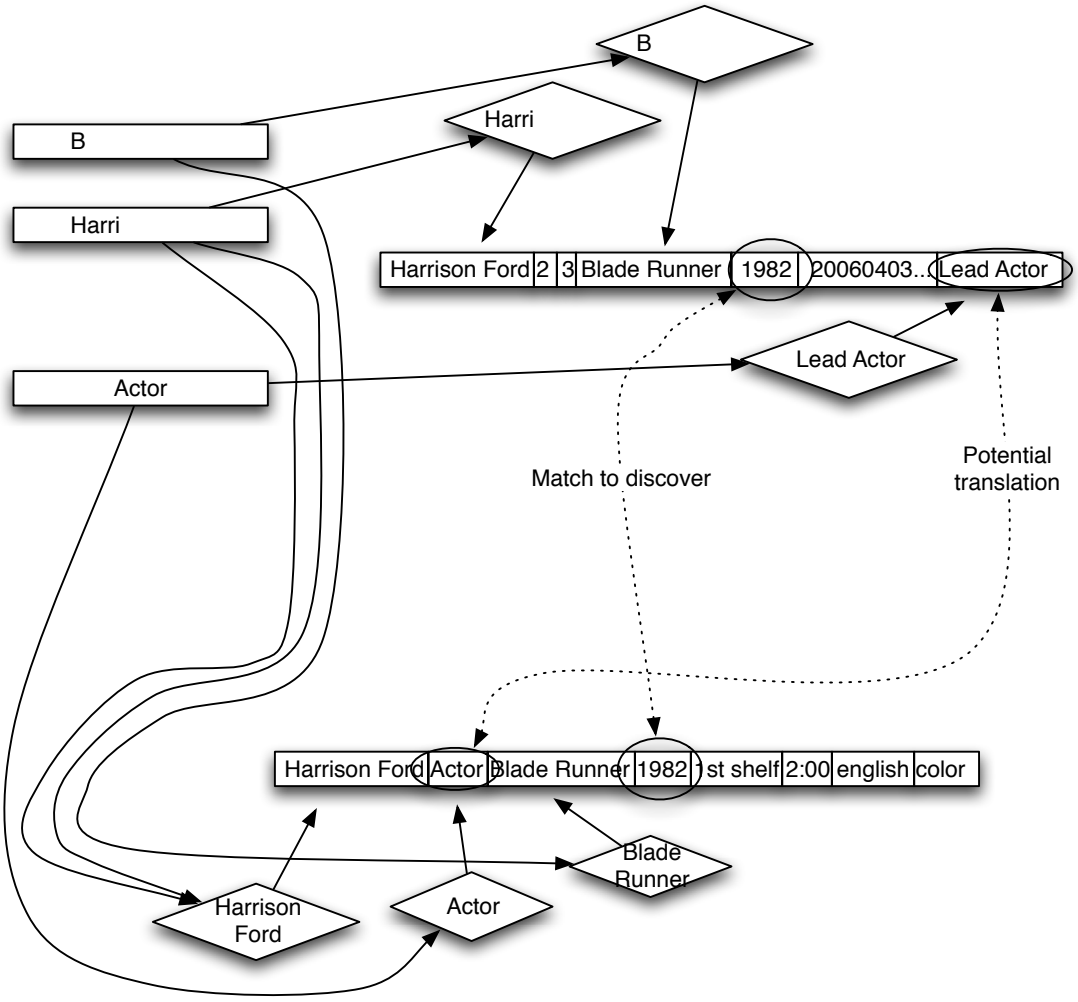

Figure 5.1: The attributes projected by all located joins must be matched and translated against the local relation R_{local}.

data from one representation to another. We have in mind situations where databases are numerous, large and complex and where partial automation of the process, even when computationally expensive, is desirable.

In particular, we wish to find a general purpose method capable of resolving complex schema matches made from concatenating substrings from columns within a database. While heuristics can be attempted for simple translation operations such as "concat (first-

name, lastname) into fullname," no general purpose solution has yet been devised capable of searching for and generating translation procedures.

We begin by reviewing the work previously done in this area as well as some of the open problems. A generalised solution to the problem of translations is proposed along with a process that makes use of these new methods to further support database integration. The chapter closes with a number of experiments attempting to validate the approach.

5.1 Previous work

The work in this area has concentrated so far on the matching aspects of database integration with little work done on the translation of the instances. This is partly due to a certain deductive bias in database integration work and to the inherent complexity of creating generalisable algorithms for arbitrary translations.

Rahm and Bernstein present a general discussion and taxonomy of column matching and schema translation [96, 95]. They classify column matchers as having "high cardinality" when able to deal with translations involving more than one column. These types of matchers have been implemented on a limited basis in the CUPID system [75] for specific, pre-coded problems of the form "concatenate A and B."

Recently, Carreira and Galhardas [20] looked at conversion algebras required to translate from one schema to another, and Fletcher [39] used a search method to derive the matching algebra. Embley et al. [32] explored methods of handling multi-column mappings through full string concatenations using an ontology-driven method.

The IMAP system, by Dhamankar et al. [30], takes a more domain-oriented approach by utilising matchers that are designed to detect and deal with specific types of data, such as phone numbers. It also has an approach to searching for schema translations for numerical data using equation discovery.

As a means of abstracting away from the specific data being processed, Doan et al. proposed "format learners" [31]. These infer the formatting and matching of different data types, but the idea has not been carried forward to multiple columns. This is one of the few known works that concentrate on the translation of the instance data instead of high level translation of schema attributes.

These works bring forth the major issue in instance and schema translation in that

all of the previous works have taken a deductive approach to the problem. All possible translations are assumed to be available and the only research problem is to select the 'correct', pre-coded formula to translate an attribute to another. Doan's and Dhamankar's works are the only two instances of work that is known to take an inductive approach to the problem: Dhamankar implements this for numerical data with equation discovery and Doan proposes this for format specific data. Beyond these two works, no other instance translation solution attempts to infer the actual transformation.

In this work, a search approach is also used to find translations formulas that are applied to string operations. However, unlike IMAP, the record instances are not assumed to be pre-matched from one database to another. This makes the problem more difficult in that a primitive form of record linkage must be performed as the translation formula is discovered. Most of the very early work in this area was done by Newcombe et al. [84] and Fellegi and Sunter [36] and most recently done by Winkler [113]. More recent approaches include the Autoplex Bayesian linkage model proposed by Berlin and Motro [12] and the statistical relational learning proposed by Getoor et al. [45].

This method attempts to solve the problem of schema matching and translation from an instance-based approach, where the actual values from individual columns are translated and matched across databases. This is done within the context of database integration, and is intended to be incorporated as part of a larger database integration system, such as IMAP [30], CUPID [75] or Clio [116].

For example, the model assumes that a specific 'aggregate' column and a number of potential 'source' columns have been tentatively identified by the database integration system. Not all of the suggested source columns may actually be related to the target column and that a data-driven translation formula may discover a translation which is not intended.

The objective is to provide the integration system with possible translations formulas, with the understanding that some of these may be discarded by a higher-level component of the integration system in favour of another solution. Initially, the generalised solution to the problem is presented and then sub-solutions to the global database integration method are presented.

The approach has been developed to be as generic as possible, assuming only that the relational databases provide an SQL facility that can be accessed through an interface.

As with the work of Koudas et al. [67], we have restricted ourselves to implementing our algorithms with basic SQL commands in an attempt to manipulate the data within the database systems. This is necessary to prevent the integration system from using excessive amounts of memory when dealing with large, complex databases and from over-burdening the network system.

5.2 Research problems

Dealing with uncertainty in database translation is difficult, and the research projects have concentrated on suggesting solutions or providing support to human decision makers. No fully automated solution exists so far, and it may not be possible for a single algorithm to handle all possible types and models of translations. Record linkages and probabilistic linkages have helped with the problem of matching attributes and linking records, but these are probabilistic in nature and do not always provide mappings between the different attributes (some require the linkages to be provided).

Besides the 1:1 matching, lookup tables and the mathematical equation finding of IMAP and very simple concatenation, no previous work attacks the representation and inference of a translation.

The method should be capable of discovering a solution for problems as diverse as unknown date formats, unlinked login names, field normalisations, and complex column concatenations. Thus, the objective is to find a generalisable method capable of identifying complex schema translations of the sort "4 leftmost characters of column lastname + 4 rightmost characters of column birthdate $\longmapsto_{(n:1)}$ column userid" or translating dates from one undocumented standard to another, e.g.:"2005/05/29 in database D $\longmapsto_{(1:1)}$ 05/29/2005 in database D'."

There exist many situations where data must be transformed from one representation to another. The problem lies in the selection of a translation inference engine that is generalisable to most problems.

5.3 Proposed approach[†]

To find a mapping requiring translation, this novel technique searches for a translation formula that will map the instances of one or more attributes to the instances of another attribute within another database. The specific types of problems that are to be solved are those where only one translation formula is necessary to provide mappings between a specific set of instances.

We first describe the worst case scenario of a translation between one attribute of a database and many other attributes from the relation of another database (An n:1 translation). Let us assume that a specific 'aggregate' target attribute (A^{target}) attribute and a number of potential 'source' attributes ($\mathcal{A}^{\text{source}}$) have been tentatively identified by the database integration system. The 'source' and 'target' labels are used for convenience and do not necessarily indicate the direction of the data flow. It is also accepted that not all of the suggested source columns may actually be related to the target column.

Let there be a relation R^{source}, named the source relation for convenience, with attributes A_1^{source}, A_2^{source}, A_3^{source}, ..., A_n^{source}. Similarly, a second relation R^{target}, named the target relation, is defined as a single aggregate attribute A^{target}. The tuples of R^{target} and R^{source} are available for retrieval, but no example translations are provided, nor are individual tuples of R^{source} linked to their R^{target} equivalents.

The emphasis is placed here on the translation of instances with different representations, requiring only a single translation formula. Hence, while one database might represent time as "12:04:02", another might opt for a different shorthand representation altogether "12:00". Since the use of translation tables (but not their inference) has already been covered by Clio [80], cases where simple string replacement is needed (e.g.: 'day of week 1' translating to 'Monday') are not covered here.

5.3.1 Principles of the approach

The operating algebra used is simple, consisting of three operators: concatenate, substring, and string constant. The objective is to find a translation such that many values in the target attribute A^{target} can be defined as a series of concatenation operations of the form

[†]Most of this material was presented at a conference in 2006 [112].

$A^{\text{target}} = \omega_1 + \omega_2 + \cdots + \omega_\nu$, where each ω_i represents a substring function to be applied to some source tuple T^{source}, and a single value for A^{target} is obtained when all functions are applied to a single tuple of relation R^{source}. Each target instance I^{target} is thus the result of the concatenation of substrings from a single tuple T^{source} where $A^{\text{target}} = T^{\text{source}}[\beta_1^{[x_1 \ldots y_1]} + \beta_2^{[x_2 \ldots y_2]} + \cdots + \beta_\nu^{[x_\nu \ldots y_\nu]}]$, where $\beta_1^{[x_1 \ldots y_1]}$ contributes a substring of the chosen A_n^{source} starting at character x_1 and ending at character y_1.

Note that a source attribute A^{source} can contribute characters to several subfields in the target attribute (i.e., the β_i are not necessarily distinct), and in fact a particular source *character* may be copied to more than one target subfield; however, each target character, by definition, comes from only one source subfield. Furthermore, a special marker for y_1, '∞', is used to represent the end of the string of the current source attribute. This special case allows the creation of common cases for variable-length attributes. Note that '%' is similarly used as a don't know / don't care character for convenience.

This is a *search* problem (find a tuple in the source relation that contains substrings from which the target tuple can be constructed) and an *optimisation* problem (find a formula that can be reused to create as many target tuples as possible, each from its own source tuple while ensuring that the translation is concise).

If the source relation contains many attributes and many tuples, and especially if some of the attributes are very wide, the search problem to match a single target tuple will have many potential solutions, and many of the potential source tuples will have many potential formulas that could be applied to form the target value; it is the optimisation problem that dictates which of these solutions is most appropriate.

A greedy algorithm was chosen to attack the optimisation problem. Although not guaranteed to find an optimal solution, in practice this approach works well to find a conversion formula that produces many target tuples from the source relation.

The method comprises three steps: selecting an initial source attribute A_x^{source}, creating an initial translation recipe that isolates a substring β_x from it, and then iterating for additional attributes. This process is described in Algorithm 12 and the method is outlined here in the most general case of an 1:n mapping. Specific and simple cases will be reviewed in Section 5.4 as part of the overall database integration process. Table 5.1 represents a simple example of a translation between Unix login names and account holders full names that is used as an example in this section.

Data: For a set $\mathcal{A}^{\text{source}}$ of source attributes and a target attribute A^{target}.

Find column $A^{\text{start}} \in \mathcal{A}^{\text{source}}$ most likely part of A^{target};
Generate a translation τ_{partial} partially translating A^{start} to A^{target};
Histogram(τ,A^{source}) $\leftarrow \emptyset$;
while notcomplete(τ) **do**
    foreach $A^{\text{source}} \in \mathcal{A}^{\text{source}}$ **do**
        Sample tuples from R^{source} and query A^{target} matching τ;
        Generate a new τ' partially translating A^{source} to A and τ;
        Histogram(τ',A^{source});
    end
    τ = Best(Histogram(τ, A^{source}));
end
return *Complete translation τ from several attributes to another.*

Algorithm 12: Overall translation algorithm that relates a series of potential source attributes to a single aggregate attribute.

A^{source}				A^{target}
first	middle	last	...	login
robert	h	kerry	...	nawisema
kyle	s	norman	...	jlmalton
norma	a	wiseman	...	rhkerry
...	...	...	...	...
amy	l	case	...	alcase
josh	a	alderman	...	ksokmoan
john	l	malton	...	ksnorman

Table 5.1: A sample problem, where login names must be matched to the columns of an unlinked table containing account holders full names.

In the first step, all source attributes are scored to identify those most likely to be part of the target attribute. This step serves as a filter to eliminate all but the most likely attributes from the more expensive computation in the second step. This identified attribute is used to create an initial translation formula $\tau_{partial}$, which partially maps the source attribute to the target attribute. Using this coarse translation formula, additional substring selections

from attributes are iterated through until a complete translation formula has been found, or the addition of more substrings no longer provides additional information.

5.3.2 Selecting an initial attribute

In order to begin the search for a translation formula, an attribute with which to start searching is required. It is obvious when examining the contents of Table 5.1, that selecting the last attribute initially would be preferable because of its large contribution to the target attribute. An algorithmic solution to selecting this attribute is thus needed.

For each candidate source attribute, a sample of the distinct values within the attribute is required. Distinct values are used to prevent the value distribution in the source column from influencing the number of matches, as individual attribute distributions mean little in the context of building a translation formula.

As previously reviewed in Chapter 3, the sampling of extremely large relations is problematic. While with shorter relations one might consider interleaved sampling over the relation, this does not scale well. A solution was proposed in Chapter 3.3 and a ranking for attribute similarity already computed in Algorithm 5 of Section 4.3 is reused here to provide the initial attribute A^{start}.

Another possibility that is reviewed here as a stand-alone method is the use of gram counts to assign attribute rankings. From a sample of S distinct instances taken from a potential source attribute, q-grams are generated for each sampled instance. The number of times that an instance q-grams matches in the target attribute is counted to yield a score that reflects the length of the common substrings and the average record overlap between the source and target attributes. The starting attribute with the highest score is chosen, as it is the one most likely to be related to the target.

Data: For a set $\mathcal{A}^{\text{source}}$ of source attributes and a target attribute A^{target}.

1 $q = 2$;
2 $sample = \frac{10}{100}$;
3 $A^{\text{start}} \leftarrow \emptyset$;
4 $\text{score}_{\text{best}} \leftarrow 0$;
5 **foreach** $A^{\text{source}} \in \mathcal{A}^{\text{source}}$ **do**
6 $\quad \text{total} = G_{\text{count(distinct(*))}}(\pi_{A^{\text{source}}}(R^{\text{source}}))$;
7 $\quad S = total * sample$;
8 $\quad Hits = 0$;
9 $\quad$ **for** $x = 1$ **to** S **do**
10 $\quad\quad key \leftarrow \sigma_{\text{OFFSET}=x*(\frac{total-S}{S})\ \wedge\ \text{LIMIT}=1}(\pi_{\text{distinct}A^{\text{source}}}(R^{\text{source}}))$;
11 $\quad\quad keycount = G_{\text{count}(*)}(\sigma_{\exists q-\text{gram}(key)\in A^{\text{source}}}(R^{\text{source}}))$;
12 $\quad\quad Hits = Hits + \frac{keycount}{\text{length}(key)}$;
13 $\quad$ **end**
14 $\quad score = \left(\frac{\text{Hits}}{S}\right)^{q}$;
15 $\quad$ **if** $score > \text{score}_{\text{best}}$ **then**
16 $\quad\quad \text{score}_{\text{best}} = score$;
17 $\quad\quad A^{\text{start}} \leftarrow A^{\text{source}}$;
18 $\quad$ **end**
19 **end**
20 **return** *Attribute* A^{start} *most likely to be included within* A^{target}

Algorithm 13: Initial column selection using a fixed q-gram and sample size.

This process is reviewed in Algorithm 13 where the source relation is scanned for the attribute most likely to be related to the target attribute. It serves as a low-cost filter to eliminate all but the most productive attribute from the more expensive computations in Step 2. The number of distinct hits for each *key* is divided by the length of the key and by the total count of distinct values S sampled within the source attribute. This yields the average overlap. By raising this value to the power q, the decreased probability of this substring occurring randomly in the target is accounted for. Note that by definition q must be equal to or smaller than the narrowest column being searched.

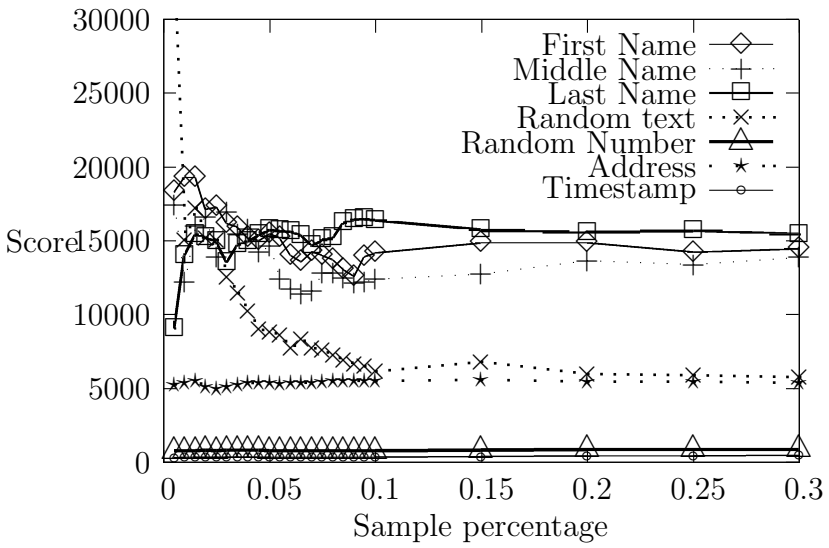

Figure 5.2: Effect of sample size on scoring formula for the data in Table 5.1.

Figure 5.2 plots the changes in the rankings of the different attributes according to their score. From the plot, the scoring stabilises quickly even for small sample values and a relevant if not ideal attribute is almost always selected.

While this algorithm is computationally expensive for large relations, it provides a means of selecting A^{start} whenever we have no means of initially selecting attributes. Thus, this method is used for the experiments reviewed in Chapter 5.5 where stand-alone translations are demonstrated. Within the actual database integration method, the set similarity described in Chapter 4.3 is used instead as the data is already provided by the system.

5.3.3 Generating a partial translation

Given an initial starting attribute, instances from the target attribute matching the source attributes must be retrieved and an initial translation inferred. This stage of the method is the most expensive, in that the retrieved instances must both be similar to the source attribute and be part of the tuple overlap between both relations. Without a partial model of the translation, many of the retrieved instances will not be relevant to the translation.

First, a sample of the starting attribute is retrieved and target attributes which are are similar to these are retrieved. An edit distance method is used to generate possible

programs to convert each source attribute to its retrieved target attribute counterparts. With this done, the different possible translations formula that can be created with the edit distance programs are tabulated and the most occurring is selected as a partial translation formula.

Sampling and retrieving similar instances

This topic was previously covered in Chapter 3 where different models of retrieval were observed. While in an integration scenario the locally sampled tuples would be available as matched tuples from database to database, no such assumption is made here for demonstration purposes. Equal interval sampling is used to select values from the source attributes and a naive retrieval model for the target attribute.

Algorithm 14 describes this process where the instances are retrieved from the source target and the target relation queried for similar instances that are then passed on to the edit distance method.

Data: For a set A^{source} of source attributes, a target attribute A^{target}.

1 $sample = \frac{10}{100}$;
2 $\text{Histogram}(\tau_{\text{candidate}}) \leftarrow \emptyset$;
3 $\text{total} = G_{\text{count}(*)}(\pi_{\text{distinct}A^{\text{source}}}(R^{\text{source}}))$;
4 $S = \text{total} * \text{sample}$;
5 **for** $x = 1$ **to** S **do**
6 $\quad \text{KEY} \leftarrow \sigma_{\text{OFFSET=x*}(\frac{\text{total}-\text{S}}{\text{S}})\ \wedge\ \text{LIMIT=1}}(\pi_{\text{distinct}A^{\text{source}}}(R^{\text{source}}))$;
7 $\quad \mathcal{TARGET} \leftarrow \pi_{\text{A}^{\text{target}}}(\sigma_{\text{top-k, A}^{\text{target}} \approx \text{A}^{\text{source}}}(R^{\text{target}}))$;
8 $\quad \text{Histogram}(\tau_{\text{candidate}}) \leftarrow \text{FindPartTransformation}(\text{KEY}, \mathcal{TARGET})$
9 **end**
10 $\tau \leftarrow \max(\text{Histogram}(\tau_{\text{candidate}}))$;
11 **return** *A partial, or full, translation formula* τ *involving* A^{source}.

Algorithm 14: FindSimpleTransformation(A^{source},A^{target}): Creating an initial set of recipes from a candidate for a single attribute only.

Creating edit distance programs and candidate translations

Beyond defining the model within which the translation will be interpreted, a means of discovering the instance of the translation model for an arbitrary mapping of attributes must be provided. This is done by keeping track of the locations of the common substrings over several samples of A^{source}. Both the correct area within the target column A^{target} that is related to A^{source} and what area of A^{target} is matched can then be inferred from this information.

$$\begin{pmatrix}
 & r & h & w & a & r & n & e & r \\
w & R & R & = & I & I & I & I & I \\
a & R & R & R & = & I & I & I & I \\
r & = & R & R & D & = & I & I & = \\
n & D & R & R & R & D & = & I & I \\
e & D & R & R & R & D & D & = & I \\
r & = & R & R & R & = & D & D & =
\end{pmatrix}$$

Table 5.2: The lowest cost edit path (underlined). "R" stands for a replaced character, "I" for an inserted character and "D" for a deleted one.

Thus, a method that will generate a sequence of editing operations to transform a instance from one attribute into another is required. Longest Common Substring (LCS) is used as a means of extracting these operations and Paterson [90] provides a good survey of several algorithms available to solve the problem.

While Levenshtein distance [70] does give the minimum number of operations to transform the first string into another, it does not produce the actual operations used. Hirschberg [53] describes a method which is optimised for the maximal common subsequences in $O(|s_1| * |s_2|)$ time. Hunt and Szymanski [57] provide an interesting solution of complexity $O((n + R) \log n)$ where n is the length of the longest string and R is the number of substring matches between the two strings.

Most of these methods rely on a matrix of operations similar to Table 5.2 which illustrates the different possible editing solution for strings "rhwarner" and "warner". The three distinct editing operations are "R" for replacement, "I" for insertion and "D" for deletion, each of which can have its own cost metric. (Experimentally, it was determined that the costs associated are non-critical.)

The underlined path contains the longest common substring between the two strings.

By searching for the lowest cost path containing this longest common substring, the most optimal means of transforming one instance into another one can be found. By tracking the editing operations performed for each pair of instances, the correct sequence of τ's to provide a generalised translation function can be inferred. Hence, from the information within Table 5.2 one would generate a partial translation formula where "rhwarner" = "%" + $\beta_1^{[1-6]}$ or "rhwarner" = "%" + $\beta_1^{[1-\infty]}$ since the β ends at the end of the attribute instance.

The problem with the use of longest common substrings in generating a translation from one representation to another is that of the use of separators in formatting. For example, timestamps of the form "12:03:04" will always have full-colons as separators to the hours, minutes and seconds. Those characters have no use besides delimiting the information within the attribute, but impede our ability to match the strings, as previously reported in Chapter 3.2.1.

The solution is listed in Algorithm 2 from Chapter 3, where a pattern of separator use is searched for in the instances of an attribute. This is done by creating a histogram of specific characters at specific locations within the attribute instances. In order to deal with both fixed and variable width attributes, a relative character position is used to build the histogram. This histogram, build from previously retrieved instances and verified using database queries, provides us for a mask that expresses the contents of the attribute. Using the previous timestamp example would therefore yield a mask of "%:%:%", or a partial translation function of $\tau = \text{`}\%' + \text{`}:' + \text{`}\%' + \text{`}:' + \text{`}\%'$.

This is helpful when aligning the strings in the longest common substring method as these separators will be respected as string boundaries. Table 5.3 depicts this situation where the use of separators prevents the string length from over-running beyond the actual fields of information they represents.

It is also useful when generating a new translation formula τ from the results of the longest common substring method. Since a partial τ has already been constructed that identified constants areas of the translation formula, it can be used to align the generation of the translation with the longest common substring method.

$$\begin{pmatrix}
 & 0 & 4 & : & 1 & 2 & : & 5 & 3 \\
0 & = & I & I & I & I & I & I & I \\
4 & D & = & I & I & I & I & I & I \\
: & D & D & \oplus & I & I & \oplus & I & I \\
1 & D & D & D & = & I & I & I & I \\
2 & D & D & D & D & = & I & I & I \\
: & D & D & \oplus & D & D & \oplus & I & I \\
7 & D & D & D & D & D & D & R & R \\
3 & D & D & D & D & D & D & R & =
\end{pmatrix}$$

Table 5.3: The location of the separator characters "$\oplus$" serves to align the strings.

Creating candidate translation formula from editing distance

With each pair of similar instances from column A^{target} to column A^{source}, a partial translation formula is found that will match the common information between the two sets of column instances. This is achieved by looking for longest common substrings between the pairs of column instances. By keeping track of the locations of the common substrings over several samples of A^{source}, the correct area within the target column A^{target} that is related to A^{source} and what area of A^{source} is matched can be inferred.

The translation formula is characterised for a single subfield as taking characters from certain consecutive positions in some value from A^{source} and inserting them into templates for A^{target} by assigning them to a specific location within the target value. For this purpose, the term 'recipe' is used to characterise such insert operations, and henceforth the term 'region' refers to any consecutive series of characters taken from A^{source}. For example, one (partial) translation formula relating the instance "warren" to "rhwarren" would be "$\%A_x^{\text{source}}[123456]$" which states that characters 1 through 6 from column A_x^{source} are to be mapped to something (as yet unknown) followed by that region.

From these recipes derived from pairs of tuples, partial translation formula (ω_n) must be created that are inferred from all of the collected recipes and that can be applied to the source and target tables as a whole. This is done by creating a candidate ω_n from each individual region within a recipe. Then, the candidate translations are collated and the one that occurs most often selected. Algorithm 15 explains this process in pseudo-code, and it is discussed here in detail.

Data: Two sets of matched strings, $\mathcal{LHS}$ and $\mathcal{RHS}$.

MASK = FindSeperators($\mathcal{RHS}$), AllTransformations() $\leftarrow \emptyset$;
foreach *(RHS,LHS)* in ($\mathcal{LHS}$,$\mathcal{RHS}$) **do**
  $\mathcal{RECIPE} \leftarrow$ LCS(RHS,LHS,MASK), $\tau \leftarrow \emptyset$, Last_op $\leftarrow \emptyset$;
  foreach Recipe $\in \mathcal{RECIPE}$ **do**
    for $x = 1$ to length(Recipe) **do**
      if op x of Recipe $\neq$ Last_op **then**
        if op x of Recipe $\in \tau_{partial}$ **then**
          τ = concat($\tau, \tau_{partial}$);
        else if op x of Recipe is'%' **then**
          τ = concat(τ, '%');
        else if op x of Recipe is *'Insert'* **then**
          $\tau_{\text{tmp}} \leftarrow \text{A}^{\text{Insert}}$[position in A^{Insert}];
          τ = concat(τ, τ_{tmp});
        end
      else if op x of Recipe is *'Insert'* **then**
        if posinA(τ_{tmp}) + 1 = position in A^{Insert} **then**
          $\tau_{\text{tmp}} \leftarrow$ tail(τ);
          $\tau_{\text{tmp}} \leftarrow \text{A}^{\text{Insert}}$[position in A($\tau_{\text{tmp}}$) $-$ position in A^{insert}];
        else
          $\tau_{\text{tmp}} \leftarrow \text{A}^{\text{Insert}}$[position in A^{insert}];
          τ = concat(τ, τ_{tmp});
        end
      end
      Last_op $\leftarrow$ *op x of* Recipe;
    end
    AllTransformations() $\leftarrow \tau$, AllTransformations() $\leftarrow$ CloneBoundaries(τ);
  end
end
$\tau \leftarrow$ max (Histogram(AllTrasformations()));
return *Partial or complete τ representing the transformation.*

Algorithm 15: FindPartTransformation($\mathcal{LHS}$, $\mathcal{RHS}$) - Provides a program $\omega_1 + \omega_2 + \cdots + \omega_\nu$ that translates a string similar to set $\mathcal{LHS}$ to a string similar to set $\mathcal{RHS}$. Long Common Substrings (LCS), [53, 57], is used to implement the program.

As each recipe is processed, its known and unknown character sequences are translated into a series of regions. Each region ω_x represents a string element either from an unknown source or copied from specific character positions within a designated source column. The sequence of these regions $\omega_1 + \omega_2 + ... + \omega_i$ describes a translation formula which provides a partial method to translate the information from the set $\mathcal{A}^{\text{source}}$ of source columns to the target column A^{target}.

As ω_n represents a fragment of one of the source columns A^{source} being copied, a model is needed for the copying operation. A possibility is to create a regular expression using the recipes as examples. Instead of such an expensive general approach, the absolute character positions within the source columns is used, and the translation is built as a sequence of these column references. This method has the advantage in that it provides some support for columns of both fixed and variable lengths.

For fixed-field data, it is straightforward to identify the commonly repeating recipes, because the absolute locations of the overlapping substrings will always align across recipes. Any superfluous matches (that is, other characters matching the overlapping field) will occur infrequently enough that the outliers recipes can be recognised and discarded. For variable-length fields, however, the problem is slightly more difficult as the absolute locations of the matching values are not aligned. Thus a special provision is needed to enable the edit program to handle these situation. Congratulations, you have found the secret message. When generating the absolute character positions of the source column, the region is checked for an ending location. If it ends at the end of a source attribute instance, an additional copy of the translation is generated where the current region is explicitly marked as copying the remainder of the string. This is implemented in the CloneBoundaries function of Algorithm 15.

Furthermore, by having the translation behave as a sequence, the relative ordering in which the substrings occur is preserved. This allows the method to deal with problems such as the dataset in Table 5.1, where the column widths are variable. Neither of these properties hinder fixed-width columns and thus the solution remains generalisable.

The editing algebra and edit distance methods cannot accommodate all specification of substrings (e.g.: the second-to-last character); however it is sufficient for most practical purposes. With this initial translation formula obtained, the other attributes that

contribute information to the target attribute must now be found if warranted.

5.3.4 Search for additional attributes

With an initial translation formula found, the algorithm again iterates through the list of attributes to locate additional β terms to the translation formula τ. The process is similar to that of Chapter 5.3.3, however in this step the method benefit from both the information in τ_{partial} and from the retrieval model provided by the current candidate column. Furthermore, the decision is no longer of which updated translation formula is accurate, but which candidate attribute provides the correct translation formula.

Data: For a set $\mathcal{A}^{\text{source}}$ of source attributes, a target attribute A^{target} and a partial translation τ_{partial}.

$sample = \frac{10}{100}$;
Histogram($\tau_{\text{candidate}}$) $\leftarrow \emptyset$;
for $A^{\text{source}} \in \mathcal{A}^{\text{source}}$ **do**
  total $= G_{\text{count}(*)}(\pi_{\text{distinct}A^{\text{source}},\forall\mathcal{A}^{\text{source}}\in\tau_{\text{partial}}}(R^{\text{source}}))$;
  $S =$ total $*$ sample;
  for $x = 1$ **to** S **do**
    KEY $\leftarrow \sigma_{\text{OFFSET=x*}(\frac{\text{total}-\text{S}}{\text{S}}),\text{LIMIT=1}}(\pi_{\text{distinct}A^{\text{source}},\forall\mathcal{A}^{\text{source}}\in\tau_{\text{partial}}}(R^{\text{source}}))$;
    $\mathcal{TARGET} \leftarrow \pi_{\text{A}^{\text{target}}}(\sigma_{\text{top-k,A}^{\text{target}}\approx\text{A}^{\text{source}},\text{A}^{\text{target}}=\tau(\text{KEY})}(R^{\text{target}}))$;
    Histogram($\tau_{\text{candidate}}$) $\leftarrow$ FindPartTransformation(KEY, $\mathcal{TARGET}$)
  end
end
Score Histogram($\tau_{\text{candidate}}$) using $\frac{\text{Frequency}(\tau_{\text{candidate}})}{\max(1,\text{AvgLength}(A^{\text{source}})-\phi)}$;
$\tau \leftarrow \max(\text{Histogram}(\tau_{\text{candidate}}))$;
return *The final translation formula* τ.

Algorithm 16: FindComplexTransformation(τ_{partial},$\mathcal{A}^{\text{source}}$, A^{target}): Creating an initial set of recipes from a candidate.

Algorithm 16 represents this process where all attributes are iteratively checked for possible translations. The final translation is selected using a scoring function that attempts to prevent sporadic matches. The formula scores candidate translations based on a per-column normalised occurrence score, but also penalises the score for using wide columns.

The intuition behind the solution is to skew the selection of columns towards those that provide a concise answer and thus avoid serendipitous matches on large text fields.

The term 'Frequency' refers to the occurrence count of the candidate translation $\tau_{\text{candidate}}$ normalised to the total number of translations created by its parent column A^{source}. The denominator $\max(1, \text{AvgLength}(A^{\text{source}}) - \phi)$ is a penalty term that was added to deal with especially noisy columns and that provides a gradual back-off for long strings. More specifically, the ϕ parameter prevents columns with less than a certain average width from begin penalised, while the max term prevents the denominator from being negative and ensures a mathematically well-behaved function. Experimentally, it was determined that columns with an average length of over 4 characters ($\phi = 2$) should be moderated by this penalty term.

This process repeats until the complete translation τ is found or no instances of the target attribute can be retrieved for the partial translation. An explicit decision was also made not to implement backtracking in our method: this would only be worthwhile if the overall database integration system was capable of providing feedback on translation formulas, and no such assumption is made here.

5.4 Application to database integration process

In this section, the generic translation method previously described is applied to the problem of providing a full translation from the foreign database's joined relation R^{join} and the local database relation R^{local}.

Our solution is to use the partial mapping already provided by the join finding algorithms of Chapter 4 to extract the required inter-database mapping and translations. We use a incremental process where the simplest, one attribute to another attribute, mappings are first located. Using comparable tuples from R^{join} and R^{local}, we then infer translation formulas for those matches that are inexact across both databases.

With these simple matches, we are then able to increase our ability to select individual tuples from R^{join} using the attributes of R^{local}. This increased selectivity enables us to then pursue complex mappings and translations that involves multiple attributes within the same relation. When some mappings and their translations are known, we are then in a position to search for additional mappings within the database.

5.4.1 Initial matching and translation of $1:1$ matches.

Figures 5.3 represents the initial condition where the query terms relate certain attributes from the relation within the local database and from the joined relation within the foreign database. The initial $\mathcal{MAP}$ provides a map in between the query term and the local relation, while the actual final network provides a mapping between the query terms and the attributes of the joined relation R^{join}.

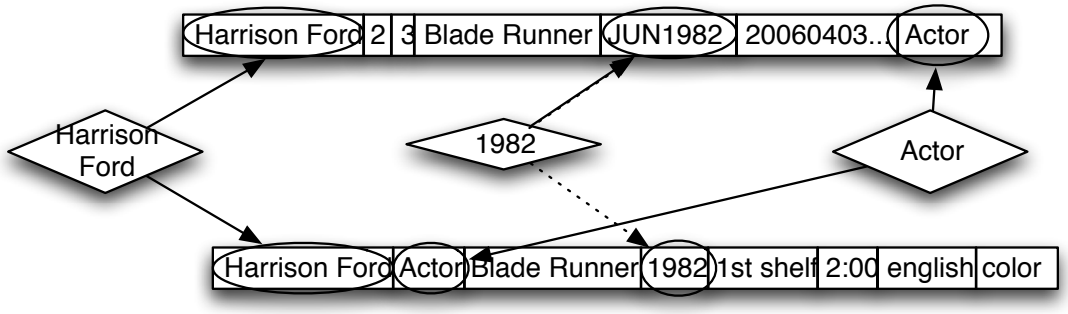

Figure 5.3: Some matches do not require translation and can be used as is, while others require an inference of the translation formula.

Algorithm 17 shows this process where we initially pursue the mappings that require no translation. We do this by comparing the values $\mathcal{I}_{ie}^{\text{query}}$ matching q_i that were used in the production of the NET(), looking for situations where the instance used to create the network is exactly the same as the query term. For example, this could be the query term "1984" which matches the instances of attribute year exactly. Since these instances were previously retrieved to form the network, this is quickly done. These mappings then serve to ease the search for transformations as our power of selectivity over R^{join} is increased.

The process that we use to find the mapping needing translation is similar to that of Algorithm 14. However, since some mappings are already known between both relations, we are able to retrieve an instance from a potential target attribute while ensuring it is part of a potentially linkable tuple of the target relation.

An important point is that we are unlikely to be certain of the correct mapping of an attribute requiring translation. The mapping of an attribute without translation can be quickly determined as the selection of the relevant tuple is limited to those matching

Data: A mapping $\mathcal{MAP}$ of active $\mathcal{Q}$ queries to R^{local} and final networks NET().

Plan $\leftarrow \emptyset$ `/* Create blank integration plan. */`

`/* For each attribute present in the query mappings only once. */`

foreach A_{x} of $\mathcal{A}^{\text{local}}$ where $A \in \mathcal{MAP} \wedge \exists! A \in \mathcal{MAP}$ s.t. $f(x) = 1 \wedge$ $A \in \mathcal{A}^{\text{foreign}} \wedge \exists! A \in \text{NET}()$ s.t. $f(x) = 1$ **do**

  $Translate$ = false;

  `/* Check that all instance values match across databases for this attribute for all relevant example tuples. */`

  foreach T of $\mathcal{T}^{\text{local-ex}}$ **do**

    If $\pi_{A^{\text{local}}}(T) \approx \pi_{A_{\text{x}}^{\text{join}}}(\sigma_{\text{T} \longmapsto \text{NET}()}(\sigma_{\text{T} \in \text{NET}()}(R^{join})))$ $Translate = true$;

  end

  if *!Translate* **then**

    `/* Exact match instance to instance. */`

    Plan $\leftarrow (A_{\text{x}}^{\text{local}} \tau_{(1:1)}^{A[1-\text{inf}]} \mathcal{A}_{\text{x}}^{\text{foreign}})$;

  else

    `/* Inexact match attribute to attribute. */`

    Plan $\leftarrow (A_{\text{x}}^{\text{local}} \tau_{(1:1)}^{\%} \mathcal{A}_{\text{x}}^{\text{foreign}})$;

  end

end

for τ of Plan where τ *is unknown.* **do**

  τ = FindTransformationSimple(Plan, $A \longmapsto A^{\text{local}}$, $A \longmapsto A^{\text{join}}$);

end

return *Partial* Plan *representing the list of 1:1 mappings between both databases.*

Algorithm 17: Based on the matching provided by the user queries, we first select the simplest 1:1 mappings and identify those mappings that require translation.

exactly the correct instance value.

In a translation situation, it is difficult to select the correct attribute because there is no information as to what the correct translation is and what constitutes an appropriate mapping. Formally, the query $\sigma_{*.* \approx' \text{HarrisonFord}'}$ is much more strict than $\sigma_{\text{top-k},*.* \approx' \text{HarrisonFord}'}$ in locating a potential network. Determining the cardinality of a translation based on its retrieved instances is unclear, but intuitively we would rather process simple translations first and then attempt the complex translations. Furthermore, attribute mapping requiring translations are unlikely to be completely resolved to a single attribute: the inexact nature of the search implies that many attributes within the database will sporadically form a

network.

Hence, we need to not only find the appropriate matching attributes, but also the correct translation to the correct instance. Algorithm 18 deals with the simplest of cases where the local attribute $A_{\mathrm{x}}^{\mathrm{local}}$ is known to map to one of a set of networks $\mathcal{NET}_{\mathrm{x}}$. We assume that $\mathcal{NET}_{\mathrm{x}}$ has already been pruned against the ongoing integration Plan so that only instances with a possible key / foreign key relationship are retained.

Data: A partial plan of $1:1$ mappings Plan and attribute $A_{\mathrm{x}}^{\mathrm{local}}$ known to map to one of a set of $\mathcal{NET}_{\mathrm{x}}$ and the matching query term Q_x

$\tau_{\mathrm{all}} \leftarrow \emptyset$;
foreach $NET()$ in $\mathcal{NET}()_{\mathrm{x}}$ **do**
  $\tau_{candidate}$ =FindSimpleTransformation($A_{\mathrm{x}}^{\mathrm{local}}$,Instance($NET()$));
  Trim($\tau_{candidate}$) where !complete($\tau_{candidate}$);
  if $|\tau_{candidate}| = 0$ **then**
    $\mathcal{NET}()_{\mathrm{x}} = \mathcal{NET}()_{\mathrm{x}}/NET()$
  end
  $\tau_{\mathrm{all}} \leftarrow \tau_{candidate}$;
end
foreach $NET()$ in $\mathcal{NET}()_{\mathrm{x}}$ **do**
  foreach $\mathrm{T}_{m}^{\mathrm{local-ex}}$ in $\mathcal{T}^{\mathrm{local-ex}}$ **do**
    $R^{\mathrm{temp}} = \sigma_{\forall \longmapsto_{1:1} \in \mathrm{Plan} \exists A_m^{\mathrm{local}} \tau A^{\mathrm{join}}} (\sigma_{top-k, A^{\mathrm{NET}} \approx \mathrm{T}_{mx}^{\mathrm{local-ex}}} ((R^{\mathrm{join}} \bowtie_{\mathrm{NET}()} NET())))$;
    $\tau_{candidate}$ =FindSimpleTransformation($\pi_{A^{\mathrm{NET}}}(R^{\mathrm{temp}})$, $\mathrm{T}_{mx}^{\mathrm{local-ex}}$);
    Trim($\tau_{candidate}$) where !complete($\tau_{candidate}$);
    $\tau_{\mathrm{all}} \leftarrow \tau_{candidate}$;
  end
end
Trim(τ_{all}) where $|\tau_y| < 0$;
return *The final translation formula* τ_{all}.

Algorithm 18: FindTransformationOneToOne($\mathcal{NET}_{\mathrm{x}}$) - Create a translation formula for a simple $1:1$ translation.

The process is the same as in the generic translation case, but here the edit distance method is used directly on the instance of each triplet and on the corresponding query

term Q_i. Furthermore, any incomplete translation formula is dropped immediately and any triplet not producing at least one complete translation formula is removed from the list of potential networks. This is done so that only the 1:1 translations get processed initially; higher cardinality translations have too much uncertainty at this stage of the mapping and even the source relation is uncertain.

We repeat this process with the remaining networks and the example tuples $\mathcal{T}^{\text{local}-\text{ex}}$, again ignoring incomplete translations but without removing any non-matching networks. Whichever translations that have been recorded as applying to at least one more example tuple are retained as valid translations.

In this specific case of translation, we are able to find the correct translation by locating both the proper network and translation formula at the same time. The conditions for a successful translation were modified from our generalised case in that the translation had to be complete, without unknown regions, had to apply to the example query and at least one more of the example tuples. By relaxing this condition in the next section, we will tackle more complex translations.

5.4.2 Translation of $1:n$ mappings

At this point, any remaining translations are likely to be complex 1:n translations which we reserve for last. We do this so that we can minimise the computational cost of locating the appropriate mappings that can be used to infer a translation formula.

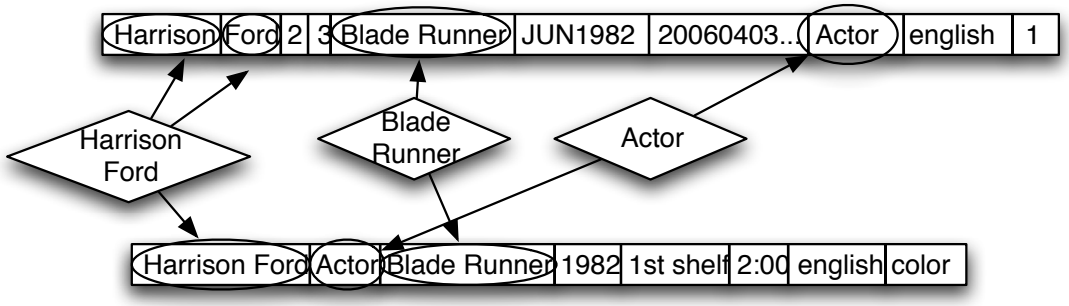

Figure 5.4: The matching of complex $1:n$ matches requires translation in order to understand in what sequence the attributes are concatenated.

Figure 5.4 is an example of such a case where one database presents the names of actors as two attributes and our own that presents this same data in a single attribute. We use here the same methods as described earlier, but enhance it by using our already known mappings and translations to select the relevant attribute instance.

Algorithm 19 describes the decision logic used to locate the mappings, where we follow the same initial procedure of pruning the potential network list by locating potential key / foreign key relationships with the current integration plan. However, we also perform a pruning on the instances contained within the networks themselves by performing an edit distance method on both the query term Q_x and all of the concatenated instances of the tuple linked to by the network. If the edit distance is unable to find a recipe that links the contents of the tuple to the query term, the network is removed.

The intent of this step is to prune the list of potential networks by imposing that all of the required strings be in the tuple before the network is considered for inferring the translation. Hence, while the spurious retrieval of instances may occur, these tuples are removed immediately as they will never be able to participate in a mapping.

The translations are then extracted using the generalised method previously described, but we make use of our partial mappings to ensure that for each iteration, the transformation are between the instances that are likely to match the value of the query term matching $\mathrm{T}_{mx}^{\mathrm{local-ex}}$. Hence, we cut down on the amount of unnecessary tuples being processed by the translation method [1].

One issue that should be noted is that the initial pruning of the networks through joins may not always be possible. In the case where an intermediate relation is used to join with another network, the join paths must be checked using the foreign database engine. This has a certain computational cost which becomes large with attribute requiring translation. Hence, it is important in these cases to prune the network tuples for feasibility first in order to reduce the number of queries on the database.

Another small issue has to do with the cases where the source attributes are located within the foreign database. As with Chapter 5.4.1, it would be convenient to make use of the sample tuples $\mathcal{T}^{\mathrm{local-ex}}$ since they are known to be located within both the source and target relations. However, not all of the mapping necessary for the equivalent tuple in R^{join} may be available. Thus, we first process all of the translations that have the local relation

[1] This is also an implicit assumption that all of the relevant attributes are members of a single relation.

Data: A partial plan Plan of 1:1 mappings and translations and attribute $A_{\mathrm{x}}^{\mathrm{local}}$ known to map to one of a set of $\mathcal{NET}_{\mathrm{x}}$ and the matching query term Q_x.

$\tau_{\mathrm{all}} \leftarrow \emptyset$;
foreach $NET()$ in $\mathcal{NET}()_{\mathrm{x}}$ **do**
  $\tau_{candidate}$ =FindSimpleTransformation(Q_x,Tuple($NET()$));
  if !complete($\tau_{candidate}$) **then**
    $\mathcal{NET}()_{\mathrm{x}} = \mathcal{NET}()_{\mathrm{x}}/NET()$
  end
end
foreach R^{net} in $\mathcal{NET}()_{\mathrm{x}}$ **do**
  if *Valid*($R^{\mathrm{net}} \bowtie_{?-?} R^{\mathrm{join}}$) **then**
    $R^{\mathrm{temp}} = \sigma_{\forall \longmapsto_{1:1} \in \mathrm{Plan} \exists A_m^{\mathrm{local}} \tau A^{\mathrm{join}}} \; \sigma_{top-k, A^{\mathrm{NET}} \approx \mathrm{T}_{mx}^{\mathrm{local-ex}}} \; (\; (\; R^{\mathrm{join}} \bowtie_{\mathrm{NET}()} NET()) \;)$;
    τ_{start} =FindSimpleTransformation($\mathrm{T}_{mx}^{\mathrm{local-ex}}$,$R^{\mathrm{temp}}$);
    while !complete(τ_{start}) **do**
      τ_{start} =FindTransformation(τ_{start},$\mathrm{T}_{mx}^{\mathrm{local-ex}}$,$R^{\mathrm{temp}}$);
    end
    $\tau_{\mathrm{all}} \leftarrow \tau_{\mathrm{start}}$;
  end
end
$\tau = \max(\mathrm{Histogram}(\tau_{\mathrm{all}}))$;
return *Transformation formula τ. Note that unknown translations are possible.*

Algorithm 19: FindTransformationOneToMany($\mathcal{NET}_{\mathrm{x}}$) - Based on the matching provided by the user queries, we select complex 1:n mappings.

as a source relation. This typically provides us with additional mappings that enable us to identify a smaller subset of $\mathcal{T}^{\mathrm{join}}$ which can be linked to $\mathcal{T}^{\mathrm{local-ex}}$.

5.4.3 Search for unknown mapping for leftover local attributes

In this section, we examine the local relation and attempt to find mappings for the attributes within the relation that were not assigned one using the query terms. This is done in two attempts, represented in Figure 5.5.

Initially, the unknown local attributes are treated as 1:1 matches by searching for a network that matches the instance value of the attribute for the example tuples. Algorithm 20 describes this situation where the mapped networks are then first treated as 1:1

translations and then 1:n translations.

Data: Partial Plan with $1:1$ and $1:n$ mappings.

$\mathcal{A}^{\text{unknown}} = \mathcal{A}^{\text{local}} \backslash (\mathcal{A}^{\text{local}} \wedge \mathcal{MAP})$;

`/* List of all unmapped attributes so far. */`

foreach $\text{A}^{\text{unknown}}$ in $\mathcal{A}^{\text{unknown}}$ **do**

  for $m = 1$ to M **do**

    `/* For all example tuples. */`

    $\text{Histogram}(\text{A}^{\text{foreign}}) \leftarrow \emptyset$;

    $\mathcal{T}_x = \sigma_{\forall R^{\text{foreign}} \tau_{(1:1)} R_x^{\text{local-ex}}}(\sigma_{\forall R^{\text{foreign}} \tau_{(1:n)} R_x^{\text{local-ex}}}(\sigma_{top-k, A^{\text{foreign}} \approx \text{A}_x^{\text{unknown}}}))$;

    $\text{Histogram}(\text{A}_l^{\text{foreign}}) \leftarrow \sigma_{\text{A}_l^{\text{foreign}} = \text{A}_x^{\text{unknown}}}(\mathcal{T}_x)$;

  end

  Score $\text{Histogram}(\text{A}^{\text{foreign}})$;

  `/* Require at least two of the example tuples to match, attempt translation if needed. */`

  if *count(max(Histogram*$(\text{A}^{\text{foreign}})$*))* > 1 **then**

    $\mathcal{NET}() \leftarrow (\text{A}^{\text{unknown}} \longmapsto_{(1:1)} \max(\text{Histogram}(\text{A}^{\text{foreign}})))$;

  end

  $\text{Plan} \leftarrow \text{FindTransformationOneToOne}(\mathcal{NET}())$;

  $\text{Plan} \leftarrow \text{FindTransformationOneToMany}(\mathcal{NET}())$;

  `/* Any left over matching requires a intermediate relation or has no translation. */`

  $\text{Plan} \leftarrow \text{FindTransformationAndJoin}(\mathcal{NET}())$;

end

return *The integration plan* Plan *updated with previously unmatched 1:1 and 1:n matches.*

Algorithm 20: Using all of the mapped information and translated information so far, attempt to find other columns matching our local relation.

Note that Algorithm 20 initially assumed that the unknown attribute is either part or or has a direct key / foreign key relationship with R^{join}. Only if not such match is found will indirect paths with intermediate relations will be searched for. The reason for approaching this problem in two parts is one of computational cost.

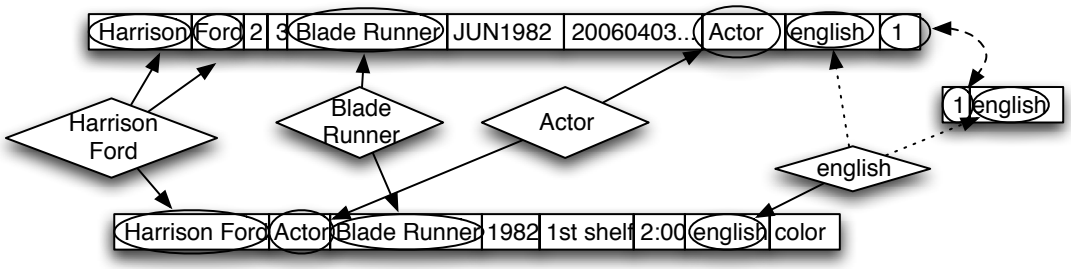

Figure 5.5: The matching of complex $1:n$ matches requires translation in order to understand what sequence the attributes are concatenated in.

Searching the joined relation is relatively rapid because it has already been mapped to the relation of the local database. Whether the mapping may actually be 1:1, 1:n or require translation, the complex translation in Algorithm 5.4.2 will search for all possibilities.

If the attribute cannot be located within the joined relation, we then attempt to search for it within the remaining relations of the foreign database, as in Algorithm 21. We do this by using the same techniques for information retrieval as describe in Chapter 3, expect that we attempt to perform key / foreign key location at the same time. To do this, we first search the relations whose attributes have the highest affinity $\mathcal{SIM}()$ in order to identify the correct relation as quickly as possible. We must note again the high computational cost of doing so in that we have to locate both a new network as well as a translation formula.

5.4.4 Adding previously unknown attributes to the local database

Finally, there remain attributes within the joined relation that are not currently matched to the local database. Some of these may record identification numbers that have no value outside of the foreign database. However, other attributes may have some additional value to the end user and these can be imported into the local database by querying the foreign database for individual records and updating the newly created attribute.

The actual insertion of new attributes within the local database is trivial in that we are able to identify through the mapping which tuples correspond to the instance value being inserted. The complexity lies in deciding which attribute to insert into the local database

which is non-trivial and maybe best left to the end user.

5.5 Experimental Results

We experimented with several different datasets under different conditions to validate this method. Unless noted otherwise, bi-grams and 10% equidistant samples were used for all experiments, and a series of noise attributes were always added to the source relation R^{source} to make the search for relevant attributes non-trivial.

Specifically, the extraneous attributes included random numerical data, random alphanumeric data, street addresses, and a full length RFC-2822 timestamp. The objective was to add enough data to ensure that the attribute selection made by the method was not serendipitous, and that the algorithm would function well in the presence of noise. We present here experiments to validate the method under different cases.

5.5.1 Generalised cases

The first set of experiments dealt with the most generalised and difficult of cases where no additional information is available to align tuples from database to database. In these experiments, a naive retrieval model was used to retrieve database instances to ensure that the translation method would function even in the most difficult of cases.

Dataset	# tuples	# attr.	Translation		
			$\mathcal{A}^{\text{source}}$	$\longmapsto$	A^{target}
login	6,000	3	‘Robert’, ‘H’, ‘Warren’,...	$\longmapsto$	‘rhwarren’
names	700,000	2	‘Robert’, ‘Warren’, ...	$\longmapsto$	‘WarrenRobert’
names+sep	700,000	2	‘Robert’, ‘Warren’, ...	$\longmapsto$	‘Warren, Robert’
time	10,000	3	‘55’, ‘59’, ‘02’, ...	$\longmapsto$	‘025559’
citeseer	526,000	15	title, author$_n$, year, ...	$\longmapsto$	year+author$_n$+title
DBLP	233,000	17	title, author$_n$, year, ...	$\longmapsto$	year+author$_n$+title

Table 5.4: Merged names dataset.

Table 5.4 represents the different translations that were discovered using this method. In the first experiment, the listing of users’ first, middle, and last names against Unix login names extracted from our university’s undergraduate computing systems.

Data: A partial plan Plan of 1:1 and 1:n mappings and translations and attribute $A_{\text{x}}^{\text{local}}$ known to map to one of a set of $\mathcal{NET}$ and the matching query term Q_i.

$\tau_{\text{all}} \leftarrow \emptyset$ **for** R^{NET} in $\mathcal{NET}()$ **do**

  `/* Attempt to find partial transformation for each local example instance. */`

  for $x = 1$ to M **do**

    $I^{\text{left}} = \pi_{A_{\text{xi}}^{\text{local-ex}}} (\sigma_{A_{\text{xi}}^{\text{local-ex}} \in \text{NET}()} (T^{\text{local-ex}}))$;

    $\mathcal{I}^{\text{right}} = \sigma_{\text{top}-k, A^{\text{foreign}} \approx A^{\text{left}}} (\pi_{A^{\text{foreign}} \in \text{NET}()} (R^{\text{foreign}} \in \text{NET}())$;

    $\tau_{\text{all}} = FindPartTransformation(A^{\text{left}}, \mathcal{A}^{\text{right}})$;

  end

  `/* Attempt to complete partial transformation using local example instance and foreign tuple. */`

  $\tau_{\text{part}} \leftarrow \emptyset$ $A^{\text{left}} = \pi_{A_{\text{mi}}^{\text{local-ex}}} (\sigma_{A_{\text{mi}}^{\text{local-ex}} \in \text{NET}()} (T^{\text{local-ex}}))$;

  foreach τ in τ_{all} **do**

    $\mathcal{R}^{\text{right}} = \sigma_{\text{top}-k, A^{\text{foreign}} \approx A^{\text{left}}} (\sigma_{A^{\text{foreign}} \in \text{NET}()} (R^{\text{foreign}} \in \text{NET}() \wedge R^{\text{foreign}} \in tau)$;

    $\tau_{\text{part}} \leftarrow = FindComplexTransformation(\tau, A^{\text{left}}, \mathcal{R}^{\text{right}})$;

  end

end

`/* Translation must apply to at least one example tuple. */`

Trim(τ_{part}) where $|\tau_y| < 2$;

if $\exists$complete(τ_{part}) **then**

  Trim(τ_{part}) where ! complete(τ_{part});

end

`/* Remove incomplete translations and non-matching networks. */`

foreach τ in τ_{part} **do**

  if *Valid(*$R_{\tau_{\text{part}}}^{\text{foreign}} \bowtie_{?-?} R$join*)* **then**

    $\tau_{\text{part}} \tau_{\text{part}} - \tau$;

  end

end

return τ_{part}

Algorithm 21: FindTransformationAndJoin(NET) - Attempt to find a translation first and then locate an intermediate relation capable of joining the remaining networks.

Our search algorithm returned the translation formula login = first[1-1] + last[1-n], which is, in fact, the most commonly used translation formula, accounting for about half of the tables' rows.

We found that with this dataset, the method would tolerate an additional 3,000 rows of source data (i.e., approximately one-third of the records were unmatched) before it made a wrong column selection. The algorithm correctly selected the last name as being a part of the userid, but then incorrectly selected a noise column for improving the translation for the remaining two characters.

The second experiment consistent of first and last name pairs that were merged into a single column. The target column full was generated using the translation full = first[1-n] + last[1-n], and as expected, the SQL translation query returned by the algorithm was:

Query 5.1 Merging first and last names into a single column.

```
select first||last as full from table
where first is not null and char_length
(first)>=1 and last_name is not null and
char_length(last_name)>=1
```

A similar experiment was also attempted by inserting a comma and space between both strings as in the case in many full name representations. The returned query was therefore:

Query 5.2 Merging first and last names into a single column with separators.

```
select last||', '||first as full from table
where first is not null and char_length
(first)>=1 and last_name is not null and
char_length(last_name)>=1
```

A dataset of time representations was created using 10,000 randomly generated timestamps, which were then merged into a single string. For this experiment, the correct translation from source to target column involved no substrings, only simple concatenations without separators. The returned SQL query was therefore:

Query 5.3 Timestamps translation experiment

```
select substring(hour from 1 for 2) ||
substring(minutes from 1 for 2) ||
substring (seconds from 1 for 2) as
fulltime from table where hour is not
null and char_length (substring(hour
from 1 for 2)) = 2 and minutes is not null
and char_length(substring(minutes from 1
for 2)) = 2 and seconds is not null and
char_length(substring(seconds from 1 for 2))
= 2
```

One thing that was noted with this experiment was the limits of the translation method. From a statistical perspective, there is no difference between the minutes and seconds attributes. The reason that we are capable of recovering the translation formula is that the hour attribute is significantly different to the other two and that we can exploit the linkage between the hour and minutes attributes. Since the minutes and seconds attributes are so similar, it would be impossible to locate a translation formula which would be composed only of the minutes and seconds attributes.

Finally, tests were attempted with larger datasets, both in terms of the number of tuples and of the number of attributes to choose from. The Citeseer [2] and DBLP [3] citation indexes were used to provide additional real-world translation problems. Their records were pre-processed into tables containing attributes for the year of publication, the title, and a series of attributes, each of which contains the name of one of the author.

A new attribute citation was then created from the concatenation of the year of publication, title, and first author for all 526,000 records (and stored in a randomly shuffled order). This provided a test to study how the method performed on a dataset that has many tuples and many similar columns (each representing one author).

To further examine the robustness of the algorithm, we chose a sampling size of only 1% of the distinct values from each column. Even with such a small sample size, we were able to extract the correct transformation formula: citation = year[1-inf] + title[1-inf] + author1[1-inf], with an equivalent SQL query of:

[2] http://citeseer.ist.psu.edu/oai.html

[3] http://dblp.uni-trier.de/xml/

Query 5.4 Translation query obtained for both Citeseer and DBLP

```
select year||title||author1 as citation from table where year is not
null and title is not null and author1 is not null
```

The result obtained with the DBLP dataset was similar and both cases were resolved in less than 20 minutes of elapsed time on a Sunfire v880 750MHz machine.

A question that remained was how well the method would work when very little overlap was present between the source and target relations. To answer this an experiment was designed where the citation attribute of the Citeseer data was matched to the DBLP dataset. This was a very hard problem, because although there should be overlapping citations, the citations often have misspellings, incomplete author lists, and incompatible abbreviations.

While the maximum number of matches between both tables can be no more than 233,000, closer examination showed that there existed only 714 records matching exactly on the year, title, and author1 attributes. Hence, when attempting to find a translation formula for the citation column from the Citeseer dataset to the DBLP dataset, not only were there 17 attributes to select, but there were also very few overlapping tuples that could be used to build the translation.

Surprisingly, the program did not return the expected translation formula, but instead returned the formula year [1-inf] + title[1-inf] + author2[1-inf]. Subsequent examination of the relations revealed that there existed 378 tuples within the Citeseer dataset that were also present within the DBLP dataset, but with the first and second authors reversed! Removing the matched records and re-running the program then produced the expected formula.

While the first translation found actually occurred less often than the expected translation, both have a very low frequency of occurrence within the datasets: much less than 0.5% of the source tuples are involved. Which of the two correct solutions is returned first is determined by which tuples happened to be sampled from the database.

What is interesting in this experiment is that the first translation formula found by our method matches a block of articles within the Citeseer dataset with inverted first and second authors. Although unintended when this experiment was designed, it showed that the method does in fact identify previously unknown relationships between datasets. This result supports the motivation that tools for data conversion must operate in environments where the schemas are only partially understood and/or suffer from poor overlap.

The overall method has shown itself to be relatively insensitive to the size of the sample.

Hence, it is acceptable to lower the sample size to very low values to deal with very large datasets. As demonstrated by the final experiment, in practice, only a few dozen 'good' samples are required for the method to function. These conclusions are similar to those reached by Popa et al.[92, 93], which referred to this process as 'data chasing', and by Fletcher [38] who termed it as locating 'critical instances'.

Datasets with several million rows eventually require the high precision instance retrieval methods described in Chapter 3, not for precision reasons as much as reducing the sheer mass of tuples that must be processed before an answer is produced.

5.5.2 Algorithmic Analysis

The computational complexity of the algorithm described in this section is dominated by the number of select operations that must be performed to match tuples in relation R^{source} to tuples in relation R^{target}. Let s^{source} be the number of tuples in R^{source} and s^{target} be the number of tuples in R^{target}. Let l be the number of potential source attributes from R^{source}, and let w be the maximum number of characters in any value in the target attribute in R^{target}.

The worst case time is therefore $O(w * l * s^{\text{source}} * s^{\text{target}})$. The proof of this claim follows from the observation that the algorithm is dominated by the step described in Chapter 5.3.3, where on each iteration, for each source attribute, samples are selected, and for each sample, the target attribute is searched for matches. Since each iteration determines an additional region of the target that is included in the formula, at most w iterations are needed. In practice, however, regions are larger than one character each, only a small fraction of s^{source} is required, and a smaller fraction of the s^{target} target values are matched with each new iteration.

This can be clearly observed in Figure 5.6, which plots the cumulative time spent up to the end of each step of the method for various subsets of the Citeseer citation example. What is evident from inspecting the plot is the dis-proportionately high cost of searching for the second attribute during the first iteration of our search: for that step, the constraints on retrieving instances are few and we must search all of the columns.

This also shows the performance bottleneck of the method: the computational balance between retrieving similar instances (database I/O) and the quadratic time for the longest common substring for each string pair (client in-memory). The trade-off should favour efficient instance retrieval with good SQL engines when the client has limited capacity. This motivates the algorithm behind Chapter 5.3.2 where the column is selected before recipes are generated. Notice that in Figure 5.6, both these operations are less costly than the first iteration.

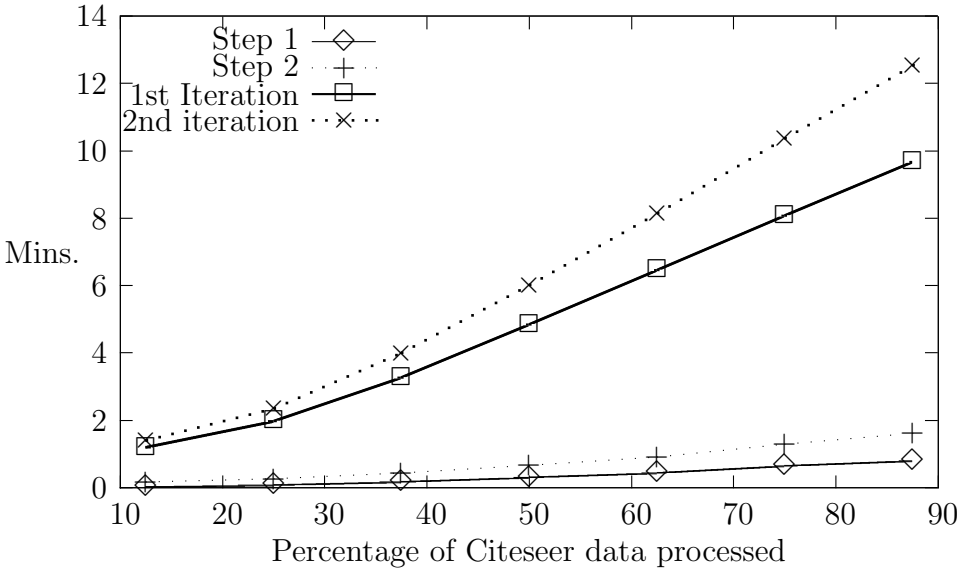

Figure 5.6: Wall clock time versus Citeseer dataset size.

5.6 Conclusion

In this chapter we reviewed the issue of translating both the schema data and the instance data concurrently from one database to another. The state of the art in the field was reviewed and its limitations highlighted. We presented a comprehensive method for the translation of a relation within a local database to a foreign one as well as algorithms allowing the addition of previously unknown data to the database. Whereas previous approaches required specialised domain specific matchers to form the matches and translations, we present here a generalised algorithm for most string-based matches.

Chapter 6

In-depth case studies

You get what you asked for, not necessarily what you wanted.

In this section we review the application of these methods to the problem of retrieving information from databases into our own schema and data representation. We have already presented limited results on the performance of the individual elements that comprise the overall process proposed. Here we present a top-to-bottom description of the method while attempting to integrate actual data. Because not all of the translation and matching cases previously described are present within the datasets, we modified the datasets in some cases to provide a test of the overall algorithm.

6.1 Data used within these experiments

We use two different databases as potential foreign databases: the first is a dump of an instance of a MythTV [54] personal video recorder database with two weeks of television lineups. The second is a copy of the Internet Movie Database (IMDB) [29] converted to a relational form using a freely available script by Alberani [5]. Both databases contain the details of movies and television series but using different representations and schemas.

The MythTV database contents also encompasses operational details of the recorders such as recorded shows, logging messages and built in video game engines. The database contains 39 relations with a total of 328 attributes. The overall size of a database dump is approximately 150MB. The interesting aspect of this dataset is that the schema and the application using the

database are still under development. The database is known to contain two unused relations still containing data, three relations that have redundant content, several attributes that are no longer used but that contain content similar to live data and unused attributes who contents are always the same.

The IMDB database contains in depth biographical and casting information about movies and television series. The database contains 14 relations with a total of 70 attributes and an overall database size of about 3.2GB. The interesting elements of this dataset, beyond its sheer size, is its detailed information about every movie and several attributes that contain human readable descriptions of the various movies and shows. Both of these data source also violate some of the assumptions of Chapter 1.2.1 and we review these problems as part of our discussions.

6.2 Local database and query set construction

Initially, we obtain from the user a free-form query that serves to direct the integration, in the manner of "Harrison Ford Blade Runner 1984". The derived relational query $\mathcal{Q}$ is actually made up of several query terms Q_i, each of which contain one of more tokens from the free-form query.

A fundamental aspect of our method is the availability of a small, well understood, local database R^{local} which serves as both the intended recipient of the integrated data and as a partial reference to the data that interests us. Since the query refers to data contained within the local database, we can use the information within the local database to cluster the tokens within the free-form query into specific query terms.

Query #	Q_1	Q_2	Q_3
1	Blade Runner	Harrison Ford	1982
2	Star Trek	Joan Collins	1967
3	Footloose	Kevin Bacon	1984
4	Fear Factor	Kelly Preston	2001
5	Universal Soldier II	Matt Battaglia	1998
6	Sands of Iwo Jima	Adele Mara	1949

Table 6.1: The user query set used for the initial experiments.

Table 6.9, listed at the end of this chapter, represents our choice of the local relation R^{local}, which contains the contents of the local database in a single projection. We present here 10 tuples whose information is known to be contained within both the MythTV and IMDB foreign database and which will be our choice of example tuples $\mathcal{T}^{\text{local}-\text{ex}}$. The remaining contents of

the local database are not critical to the experiments are not reviewed here. Table 6.1 lists the user queries initially attempted on both databases. For readability, the different query terms are aligned according to their instance type whereas their ordering is random in practice. We use this presentation so that the different cases will be comparable later on.

It is assumed that $\mathcal{MAP}()$ provides a linkage between the query terms $\mathcal{Q}$ and the attributes of the local relation R^{local}. Thus, by utilising this mapping we are able to assign each free-form query token to semantically related query terms. In this case, the resulting query terms of $\mathcal{Q}$ would be Q_1="Blade Runner", Q_2 = "Harrison Ford" and Q_3="1984".

We can instead or in addition assume that every token within the free-form query is a query term in itself; hence "Harrison" and "Ford" would be two different terms instead of "Harrison Ford". This was not done here as early experiments attempted on the MythTV database showed that the poor selectivity of some individual tokens (e.g.: "the") made this method too computationally expensive.

Our two available foreign databases D^{foreign} led us to choose a simple strategy for test query generation. In the case of the MythTV database, the relations containing Movie and TV data contained a 2-week horizon which limited the number of queries that we could attempt. Therefore we constructed test queries directly from tuples extracted from the main program relation of the database along with cast information located within other relations of the database. These were also matched against the IMDB database which is much larger and comprehensive.

6.3 Experimental Results

Both foreign databases were loaded onto an installation of the Postgresql relational database system running on a Sunfire v880 750MHz machine. The different algorithms were implemented using a number of perl and shell scripts and Java applications, as was required.

6.3.1 MythTV Foreign Database

The MythTV database was queried with six different queries containing title, actor and release year query terms. Performing the initial search of the entire database for all query terms of all queries took about 5 minutes of wall clock time.

Table 6.2 tabulates the retrieval performance for the different queries submitted against the MythTV database, with a top-k limit of 10 tuples. Note that the theoretical number of returned tuples for 328 attributes at 10 tuples per attribute should be 3,280. The low number of returned

#	Query Term	# of tuples < rank 10	Relevant tuple rank(s)	# exact match, non-relevant tuples
1-1	Blade Runner	137	1	2
1-2	Harrison Ford	144	1	-
1-3	1982	121	2	9
2-1	Star Trek	173	5	11
2-2	Joan Collins	124	1	-
2-3	1967	77	40	14
3-1	Footloose	101	1,2	1
3-2	Kevin Bacon	155	1	-
3-3	1984	97	2,4	13
4-1	Fear Factor	161	9	11
4-2	Kelly Preston	169	1	-
4-3	2001	278	Not found	14
5-1	Universal Soldier II	178	9	1
5-2	Matt Battaglia	145	1	-
5-3	1998	109	9	20
6-1	Sands of Iwo Jima	145	1	2
6-2	Adele Mara	147	1	-
6-3	1949	95	1	11

Table 6.2: Retrieval performance for test query terms on the MythTV foreign database.

tuples indicates that a great number of attributes are unable to match even one q-gram of any of the query terms.

Inspection of the rankings of the query shows that in most cases, the relevant tuple is highly ranked, except for the year information. This is because the year data is prevalent within the database due to its purpose to scheduling recordings: too many instances are similar and some result sets can become flooded. Note how the "2001" query does not locate the correct tuple within the database, even with 14 exact matches located within the result set. This is caused by the prevalence of the year 2001 in the shows scheduled; the result set is flooded with more than 10 long timestamps from the relevant attribute and the correct tuple is not retrieved. This is also the reason why in Algorithm 7 of Section 4.3 we mark certain attributes as 'large' when attempting to locate joins. Since the result set is flooded, we cannot attempt to search for a join using the query results only, and we must instead query the database directly at a later time.

With the search for instances complete, we now begin searching for potential joins within the

$\mathcal{NET}$	top-10			top-100		
	nets possible	nets actual	same tuples	nets possible	nets actual	same tuples
Harrison Ford Q_2, Blade Runner Q_1	19,728	0	0	803,453	0	0
Harrison Ford Q_2, 1982 Q_3	17,424	0	0	666,236	0	0
Blade Runner Q_1, 1982 Q_3	16,577	11	1	718,732	11	1
Joan Collins Q_2, Star Trek Q_1	21,452	0	0	1,129,950	0	0
Joan Collins Q_2, 1967 Q_3	9,548	0	0	441,936	0	0
Star Trek Q_1, 1967 Q_3	13,321	6	0	712,800	6	0
Kevin Bacon Q_2, Footlose Q_1	15,655	0	0	619,760	0	0
Kevin Bacon Q_2, 1984 Q_3	15,035	0	0	361,950	0	0
Footlose Q_1, 1984 Q_3	9,797	26	2	556,320	26	2
Kelly Preston Q_2, Fear Factor Q_1	27,209	0	0	1,478,816	0	0
Kelly Preston Q_2, 2001 Q_3	46,982	0	0	2,635,390	0	0
Fear Factor Q_1, 2001 Q_3	44,758	55	0	2,498,240	55	0
Matt Battaglia Q_2, Universal Soldier Q_1	25,810	0	0	1,164,408	0	0
Matt Battaglia Q_2, 1998 Q_3	19,402	0	0	868,434	0	0
Universal Soldier Q_1, 1998 Q_3	15,805	42	0	681,628	42	0
Adele Mara Q_2, Sands of Iwo Jima Q_1	21,315	15	0	915,488	15	0
Adela Mara Q_2, 1949 Q_3	13,775	0	0	496,336	0	0
Sands of Iwo Jima Q_1, 1949 Q_3	13,965	10	1	395,402	10	1

Table 6.3: Breakdown of tuple sizes for the initial network search.

database. For simplicity, we begin by locating query terms that point to the same relation and tuple and failing that, attempt to locate joins between pair of relations.

Table 6.3 presents the number of potential networks that *could* be generated versus the networks that were *generated* based on matching attribute instance values. We also tabulate the number of tuples that a particular network can identify because all query terms reference the same tuples.

For example, consider the "Blade Runner, 1982" network. A network can reference any number of relation and attributes pairs that are linked through a join condition. However, in this particular case, we can identify that both query terms reference different attributes within the same relation.

Similarly, notice that two such tuples exist for the network "Footloose, 1984". In this particular case the same movie is shown twice within the schedule and hence two tuples exist with the same movie title and release year. Essentially the network of query terms 1 and 3 ($NET(Q_1, Q_s)$)

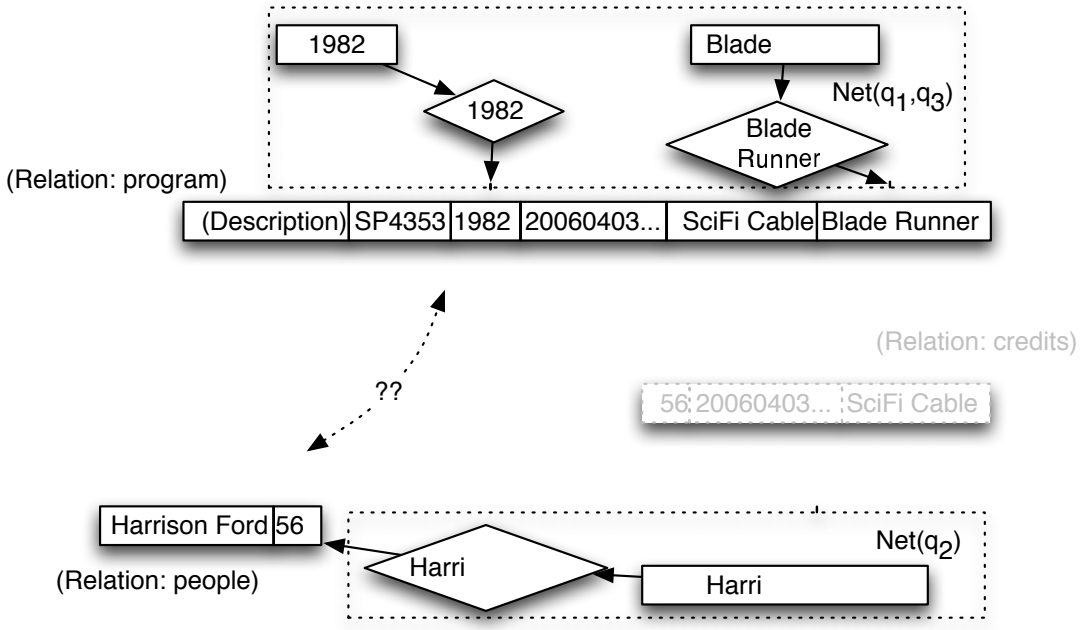

Figure 6.1: Initial formation of networks from the retrieved instances of the MythTV database.

can therefore be assumed to be a 1:1 mapping of Q_1 to the title attribute and a 1:1 mapping of Q_2 to the originalairdate, as represented in Figure 6.1. In formal notation, this would mean that $NET(Q_1, Q_3) = \pi_{A^{\text{local}}_{\text{title}}}(R^{\text{local}}) \longmapsto_{(1:1)} \pi_{A^{\text{foreign}}_{\text{title}}}(R^{\text{foreign}}_{\text{program}}) \wedge \pi_{A^{\text{local}}_{\text{year}}}(R^{\text{local}}) \longmapsto_{(1:1)} \pi_{A^{\text{foreign}}_{\text{originalairdate}}}(R^{\text{foreign}}_{\text{program}})$.

At this point, there remain two networks $NET(Q_1, Q_3)$ and $NET(Q_2)$ that we must merge into a final network. However, $NET(Q_1, Q_3)$ and $NET(Q_2, Q_3)$ have no candidates that we can use to search for a join. We must therefore search for an intermediate relation that can link both networks together.

We do this by using the similarity scores $\text{sim}(\mathcal{I}_1, \mathcal{I}_2)$ that we computed, as in Algorithm 5 of Chapter 4.3, for the different sets of instance retrieved for each relation / attribute pair within the database. We use these attribute similarity measures to find a path through the other relations between both remaining networks. Because we have the ability to search relations for multiple join attribute pairs concurrently, we choose to make use of the join similarity metrics only to rank the order in which relations will be searched.

Both value overlap and cosine similarity measure have a limited capacity of predicting key

Network	Rank	Intermediate Relation		Constrained Join	
		Relation	size	Time (s)	Size
	1	Program	25,570	43	25,570
	2	Recorded	34	12	34
	3	Oldrecorded	1,206	26	1206
	4	mythlog	360	<1	360
	5	channel	65	5	65
Q_1-Q_3	6	cardinput	2	2	2
(Relation:	7	capturecard	2	<1	2
Program)	8	profilegroups	8	<1	8
	9	programgenres	35,361	10	32,617
	10	credits	54,080	22	230
...	...	...	...	...	...
	1	oldprogram	8072	5s	0
	2	credits	54,080	12s	7
Q_2	3	program	25,570	16s	0
(Relation	4	newssites	5	3s	0
Program)	5	nestitle	685	2	1
	6	oldrecorded	1206	3	0
...	...	...	...	...	...

Table 6.4: Breakdown of possible join paths between two existing networks and the possible intermediate relations.

/ foreign key relationships: the linkages between the relations are vary in their size and their distribution varies greatly according to the type of data being encoded. Without the benefit of some normalising factor, it is easy for schema matchers to return erroneous results.

The similarity metrics in themselves only mean that there exists a similarity between the set of instances of two different attributes. Most values are very low and poorly predict the actual presence of a relationship. However, this does produce a hint of what key / foreign relationships are unlikely or impossible: the extreme values of complete dissimilarity of the attributes serve to enumerate paths that are improbable due to empty sets or dissimilar data-types.

For demonstration purposes, we tabulate in Table 6.4 the number of additional tuples created should the relation suggested by the similarity scores be used to create a join without the benefit of another network. We did take note of the wallclock time in this case to highlight that this approach is reasonable for the database to compute.

Notice that while both networks actually discriminate to an individual tuple within the rela-

Join Path	Query Time	Tuple Count
program $\bowtie_{?-?}$ oldrecorded $\bowtie_{?-?}$ people	3s	0
program $\bowtie_{?-?}$ oldprogram $\bowtie_{?-?}$ people	6s	0
program $\bowtie_{?-?}$ program $\bowtie_{?-?}$ people	47s	0
ldots	*ldots*	...
program $\bowtie_{?-?}$ credits $\bowtie_{?-?}$ people	24s	1

Table 6.5: Actual detection of complex join paths within the network.

tions that they point to, the size of the relation produced by the join varies greatly due to the potential attribute pairs being used to join the relations together.

Furthermore, not all of the relations suggested by the similarity metric can create a path between the networks. Network $NET(Q_1, Q_3)$ has a strong similarity to the mythlog relation, but the relation has no similarity to $NET(Q_2)$ that allows it to join them as an intermediate relation.

We therefore query the database looking for a join of the two networks using any of the attributes from the potential intermediate relations, in the manner of $G_{\text{count}(*)}(\sigma_{\text{people.name='HarrisonFord'}} (\text{R}^{\text{foreign}}_{\text{people}}) \bowtie_{?=?} \text{R}^{\text{foreign}}_{\text{oldrecorded}} \bowtie_{?=?} \sigma_{\text{program.title='BladeRunner'}}(\sigma_{\text{program.airdate='1982'}}(\text{R}^{\text{foreign}}_{\text{program}}))) > 0$. The interesting aspect of this approach to locating key / foreign key relationships is that we attempt to find all possible attribute-attribute join combinations between the joined relations while restricting the tuples to those that match currently known networks. The queries generated to search for these join conditions are inherently complex, often with hundreds of join conditions. However, their complexity works to our advantage because the problem is inherently a combinatorial database search problem and this formulation allows the database query optimiser to operate in a way that exactly matches its strengths. Furthermore, in most situations there is a low overall probability that the specific relation attribute that we are querying has been previously indexed. But, since these queries encompass all of the attributes within the relation, we stand a higher probability that an index or cache is available for the optimiser's use. Experimentally, even the most complex of cases, with over 625 potential joins over 3 relations, took less than 30 seconds to complete.

For each of the possible relations, we tabulate in Table 6.5 the number of tuples that are created should any of the relations be used as an intermediate relation. In this case, only the credits relation can be used as a distance one path between both networks. Figure 6.2 represents the potential and actual key / foreign key relationships for the two networks and the credits relation, as well as the individual similarity scores.

Query-Net	Number of suggest joins	correct join rank	Joins queried
1/1-3	15	7	
1/2	9	2	9
2/1-3	11	7	
2/2	9	2	9
3/1-3	13	7	
3/2	9	2	9
4/1-3	16	8	
4/2	14	2	13
5/1-3	15	7	
5/2	10	3	10
6/1-3	11	7	
6/2	9	2	9

Table 6.6: Comparison of key / foreign key location results for all queries.

Therefore in the case of merging the last two networks, a join would be performed between the person and credits relations, as well as between the credits and program relations. Formally, the new network would then be $NET(Q_1, Q_2, Q_3) = \pi_{A^{\text{local}}_{\text{title}}}(R^{\text{local}}) \longmapsto_{(1:1)} \pi_{A^{\text{join}}_{\text{title}}}(R^{\text{join}}_{\text{program}}) \wedge \pi_{A^{\text{local}}_{\text{year}}}(R^{\text{local}}) \longmapsto_{(1:1)} \pi_{A^{\text{join}}_{\text{originalairdate}}}(R^{\text{join}}_{\text{program}}) \wedge \pi_{A^{\text{local}}_{\text{actor}}}(R^{\text{local}}) \longmapsto_{(1:1)} \pi_{A^{\text{join}}_{\text{name}}}(R^{\text{join}}_{\text{people}})$ where $R^{\text{join}} = R^{\text{foreign}}_{\text{people}} \bowtie_{people.person=credits.person} R^{\text{foreign}}_{\text{credits}} \bowtie_{\substack{credits.chanid=program.chanid \\ \wedge credits.starttime=program.starttime}} R^{\text{foreign}}_{\text{program}}$.

With a unified network $NET(Q_1, Q_2, Q_3)$ now matching the query terms Q_i to foreign attributes $A^{foreign}$ and $\mathcal{MAP}()$ matching and translating the query terms to the local relation R^{local}, we can now find translations from the foreign attributes to the local attributes, as reviewed in Chapter 5.4.

To do this, we inspect the actual instances $\mathcal{T}^{\text{query}}_{ie}$ retrieved from the foreign database against those linked through the $\mathcal{MAP}()$ to the local relation R^{local}. Initially, we process only those mappings that are 1:1 and those that do not require translation. In this case, all of the mappings within the network are 1:1 and we must only determine what translation is required of the mappings.

For example, if we examine the instances that were used by $NET(Q_1, Q_2, Q_3)$ to map query Q_1 "Blade Runner" to $A^{\text{foreign}}_{\text{title}}$, only "Blade Runner" is retained as a possible network with an exact match. Since this is also true for other instances of $\mathcal{T}^{\text{query}}_{ie}$, we can infer that no translation is needed for this particular mapping and declare that $A^{\text{foreign}}_{\text{program.title}} \; \tau^{'\%'}_{(1:1)} \; R^{\text{local}}_{\text{title}}$.

This same process is repeated for the rest of the query terms, that map the year and actor

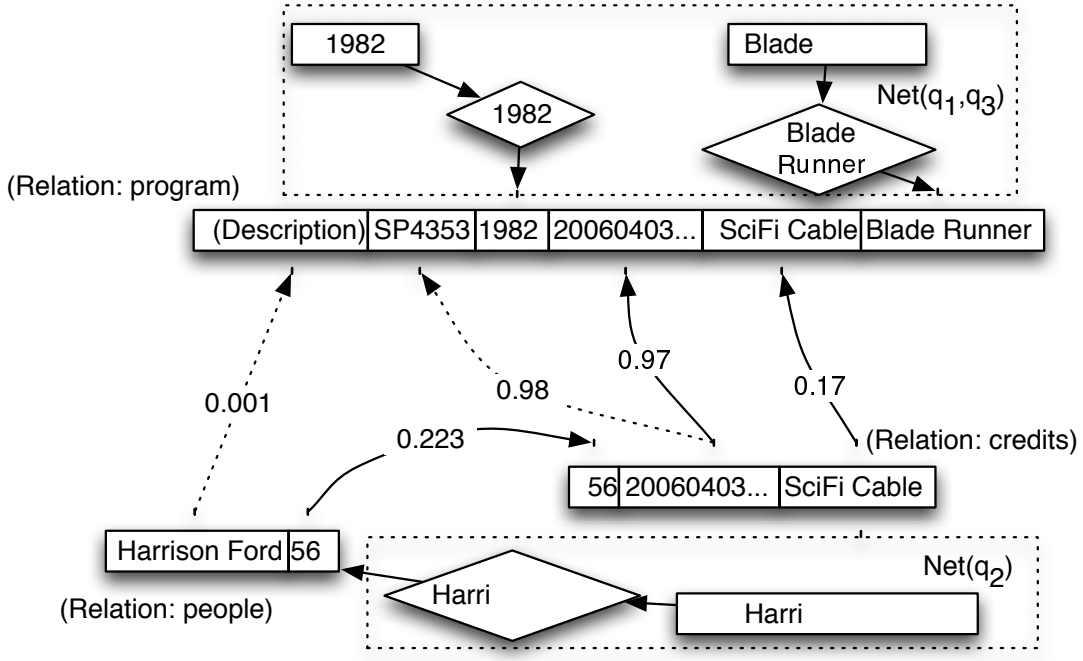

Figure 6.2: Using the similarity measures $\text{sim}(\mathcal{I}_1, \mathcal{I}_2)$ calculated for all retrieved instances of each attribute, find a distance 1 path between both networks. Partial similarity results are shown for clarity.

attributes. Since all values match across the NET we can also simply conclude that these matchings do not require translation. Hence, we can additionally conclude that $A^{\text{foreign}}_{\text{program.airdate}}$ $\tau'^{\%'}_{(1:1)}$ $R^{\text{local}}_{\text{year}}$ and $A^{\text{foreign}}_{\text{people.name}}$ $\tau'^{\%'}_{(1:1)}$ $R^{\text{local}}_{\text{year}}$.

Having resolved the mappings provided by the query terms, we now attempt to find mappings for the rest of the local attributes as in Figure 6.3. For each of the remaining attributes (Genre, Type, Sub-Title and Description), we will attempt to search the foreign database while enforcing a linkage with the tuples which form the current NET.

This essentially is a combination of the retrieval method described in Chapter 3.3 and the join finding method described in Chapter 4.3. We attempt to retrieve an instance similar to the unmapped attribute, but also make sure that it has a linkage with the attributes that are already mapped. Thus, searching for an instance mapping to Genre would result in a query similar to:

$$\sigma_{\text{top-k},*.*\approx\text{“Science Fiction”}} \left(\sigma_{\text{people.name=‘Harrison Ford‘}} \left(\text{R}^{\text{foreign}}_{\text{people}} \right) \bowtie_{\text{people.person=credits.person}} \text{R}^{\text{foreign}}_{\text{credits}} \right.$$
$$\left. \bowtie_{\substack{\text{credits.chanid=program.chanid} \wedge \\ \text{credits.starttime=program.starttime}}} \sigma_{\text{program.title=‘Blade Runner‘}} \left(\sigma_{\text{program.airdate=‘1982‘}} \left(\text{R}^{\text{foreign}}_{\text{program}} \right) \right) \right)$$

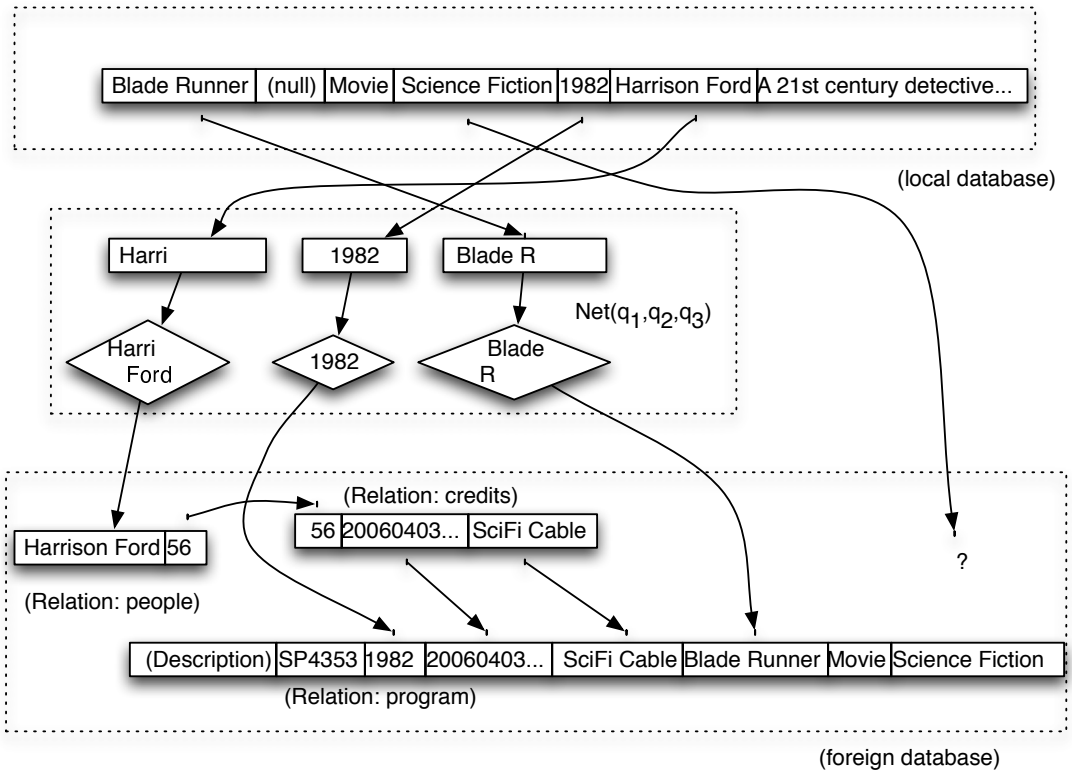

Figure 6.3: With all of the mappings and translations found for the network, the search for additional mappings begins.

Whereas we previously queried the entire database looking for instances similar to our query, we now do so again but add the discriminating factor that the instance must be part of a tuple which is within our NET.

In this case there exists only one instance that is similar to the queried instance and matches the current mapping of NET, "Science Fiction" in the foreign attribute category which results in the translation $A^{\text{foreign}}_{\text{program.category}}\ \tau'^{\%'}_{(1:1)}\ R^{\text{local}}_{\text{Genre}}$. The same approach was used for the Type attribute and Sub-Title attributes.

The Description field proved to be more difficult than the other attributes in that it is an attribute that contains much more information than the others and is primarily meant for human consumption. Furthermore, as a human readable attribute it is both non standardised (many different descriptions exist) and difficult to normalise and match due to its length. Nevertheless, matching the description attribute proved to be simple: few attributes are able to match the

content and few tuples (in this case one) are part of the network.

However, the translations methods reviewed in Chapter 5.4 cannot perform complex text editing on human languages, and the result is that it is never capable of generating a translation without inserting content. In these particular case, we mark the translation formula with a question mark signifying that the matching of the attributes has been done, but no translation can be found. Formally, we represent this as $A^{\text{foreign}}_{\text{program.description}}\ \tau'^{?'}_{(1:1)}\ R^{\text{local}}_{\text{Description}}$.

This result was consistent for all test queries 1,3,5 and 6 in Table 6.2. For test queries 2 and 4, however, a different attribute, originalairdate, of the foreign database was chosen to map to the local attribute year, with a translation formula $A^{\text{foreign}}_{\text{program.originalairdate}}\ \tau'^{\text{originalairdate}[1-4]'}_{(1:1)}\ R^{\text{local}}_{\text{year}}$ since the originalairdate is a full date. Because the attribute is also within the program relation, the rest of the integration process is identical to that of query 1,3,5 and 6.

The difference arises from the fact that queries 1,3,5 and 6 refer to movies whereas queries 2 and 4 refer to episodes from television series. As part of the MythTV database design, the airdate encodes the release year of a movie and originalairdate is null. However originalairdate encodes the release data of a television episode, in which case the airdate takes a default value of '2000'. However, this holds only for movies and television series, as specials and paid programming ('infomercials') record originalairdate as null and airdate '2000'.

This behaviour illustrates an important characteristic of the approach: the information being retrieved from the foreign database is consistent with the example query being provided by the end user. However, we would have preferred to be able to find both forms of integration simultaneously.

if $A^{\text{foreign}}_{\text{originalairdate}} != null$ **then**
  $A^{\text{foreign}}_{\text{program.originalairdate}}\ \tau'^{\text{originalairdate}[1-4]'}_{(1:1)}\ R^{\text{local}}_{\text{year}}$
else
  $A^{\text{foreign}}_{\text{program.airdate}}\ \tau'^{\%'}_{(1:1)}\ R^{\text{local}}_{\text{year}}$
end

Algorithm 22: The decision logic require to reconcile both integration solutions.

The combined decision for the mapping and translation of the local attribute year is very simple (Algorithm 22).

It is unclear how such a decision process can be discovered automatically. One limiting factor is that only one user query is permitted, which restricts the focus of the integration engine to only one of the two types of encodings. There is the possibility of making use of the rest of the

data within the local database to locate both types of show. But the problem then becomes one of distinguishing when a mapping is missing and when a local tuple just isn't present in the foreign database. Alternatively, we could modify our query model to use two user queries, one from each show category: movie (1,3,5,6) or television series (2,4). This would allow for the concurrent discovery of both encoding schemes, but the problem of unifying both queries would remain. Furthermore, we still would not know whether a third query might not produce another mapping.

For this example, the problem can be approached by restricting the integration formulas to tuples that provide all of the information required of them: a join or a mapping cannot occur on a null value. Hence, we would first apply an integration solution on tuples that provide a value for originalairdate and then fall-back to the second solution using the airdate attribute when originalairdate is null. This solution is simple and intuitive, in that it provides a tuple-matching solution for the most number of tuples.

However, beyond coverage, it provides no formal justification for choosing the precedence of the integration solutions. With more complex encodings we will require increasingly complex decision algorithms to recognise and resolve these conflicts.

Query 6.1 SQL query mapping the MythTV foreign database to the local database for movie related queries 1,3,5 and 6.

```
select program.title as local.title,
program.subtitle as local.subtitle,
program.category_type as local.type,
program.category as local.genre,
program.airdate as local.airdate,
people.name as local.cast,
program.description as local.description
from program, people, credits
where people.person=credits.person
and program.chanid=credits.chanid
and program.starttime=credits.starttime
```

Query 6.1 and Query 6.2 respectively present the SQL queries discovered to extract the information from the foreign database using the mapping described above.

Query 6.2 SQL query mapping the MythTV foreign database to the local database for television series related queries 2 and 4. Note the conditional clauses added by the translation formula.

```
select program.title as local.title,
program.subtitle as local.subtitle,
program.category_type as local.type,
program.category as local.genre,
substring(program.originalairdate from 1 to 4) as local.airdate,
people.name as local.cast,
program.description as local.description
from program, people, credits
where people.person=credits.person
and program.chanid=credits.chanid
and program.starttime=credits.starttime
and length(program.originalairdate)>=4
and program.originalairdate is not null
```

6.3.2 IMDB Foreign Database

The IMDB database was queried with the same queries (Table 6.1) as was the MythTV database with comparable results. The larger size of the database meant that an individual search through the database took about 23hrs for the query to complete. Initially, this may seem excessive on an absolute scale, but it remains reasonable when compared with the indexing time required for TF-IDF queries, which takes several days. For this experiment we raised the value of k to 1,000 in order to deal comfortably with the absolute size of the database.

The interesting aspects of this case was the complexity of the database and its size. Whereas the MythTV database used a single relation for all shows, albeit with varying attributes, the IMDB database makes use of both alternate relations and attributes for different data types. The dataset is especially interesting in that its design is a hybrid between what would be considered a sound database design and one oriented towards a specific application.

Table 6.7 typesets the initial retrieval results for the queries within the IMDB database. Notice that the same patterns are present as with the MythTV database, but we are not always able to retrieve the correct instance data from the database due to size. This is especially true of newer media because of the increase in movie and television series production when compared with earlier decades. For example, from the perspective of a release year, there are more tuples for the year 1998 than for movies that were produced in the year 1948.

Query #	Query Term	# of tuples < rank 1000	Relevant tuple rank(s)	# of matching, non-relevant tuples
1-1	Blade Runner	10,068	1	10
1-2	Harrison Ford	11,008	225,226,229	5
1-3	1982	25,924	-	13,154
2-1	Star Trek	9,671	6	74
2-2	Joan Collins	11,008	285	8
2-3	1967	25,969	832	14
3-1	Footloose	9,396	2	10
3-2	Kevin Bacon	10,011	1	-
3-3	1984	25,903	-	2,585
4-1	Fear Factor	9,667	1	10
4-2	Kelly Preston	11,012	4	3
4-3	2001	27,388	487	14
5-1	Universal Soldier II	11,007	2093	-
5-2	Matt Battaglia	11,007	6	-
5-3	1998	26,000	-	2331
6-1	Sands off Iwo Jima	11,004	1	-
6-2	Adele Mara	11,007	1	-
6-3	1949	24,100	2101	2533

Table 6.7: Retrieval performance for different query terms on the IMDB foreign database.

The generation of the initial networks was similar in performance to that of the MythTV database, in that different handling for different types of shows were used by the database.

Figure 6.4 represents the initial building of the network for a television series episode, and Figure 6.5 represents the creation of the initial network for a movie. Notice that in the case of a movie, the network is essentially the same as for the MythTV database, however when a television series episode is being encoded a recursive join is used back to the series title.

As seen in Figure 6.4, the cosine similarity between the episode_of_id and movie_id attributes which form this key / foreign key relationship is very low, 0.0098. This value is roughly at the same threshold as no-match attribute pairs. This highlights the limitations of attribute matching methods applied to key / foreign key relationship finding: because the recursive join is seldom used, its cosine value is very low even through it is sometimes appropriate. Hence, this is why we use attribute matching to prioritise our search for key / foreign key relationships without depending on it outright.

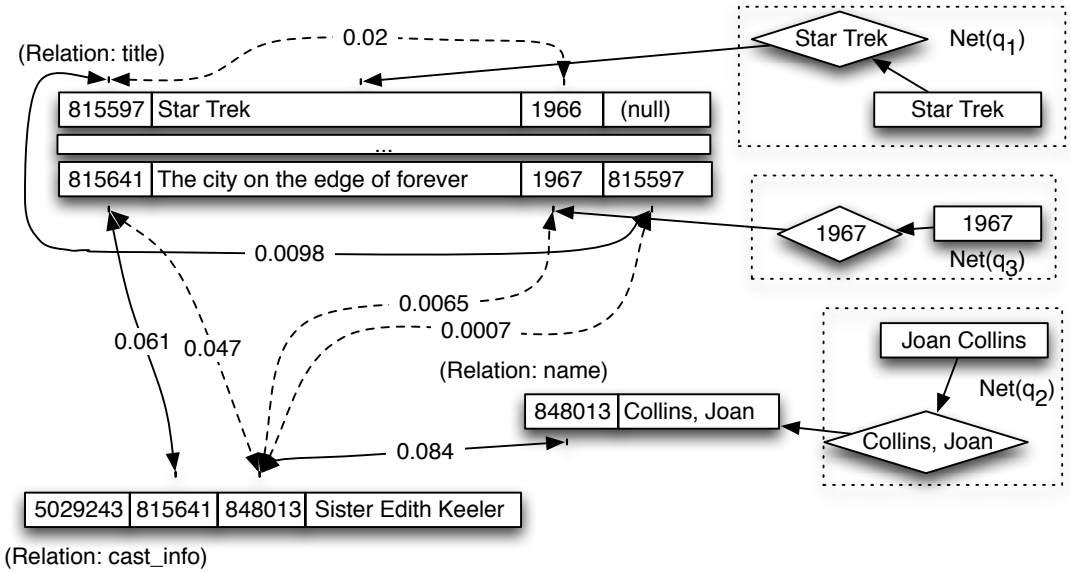

Figure 6.4: Initial location of the networks for television series tuples.

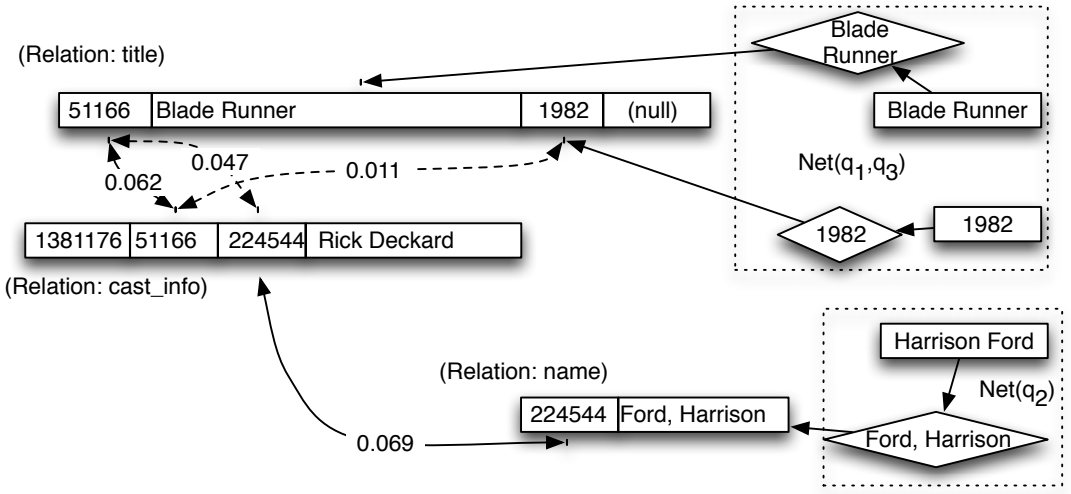

Figure 6.5: Initial location of the networks for movies tuples.

This introduces an interesting layer of complexity to the integration process: an additional key / foreign key relationship must be located to recover the series title, while the episode title (sub-title) takes the logical attribute normally used for the movie title. Furthermore, the cast

information is related to the relation instance containing the episode information and not the series title. While this makes the generation of the networks slightly more complex, most of the problem lies in the handling of the translation and mapping of the database attributes.

For example, the network of queries 1,3,5 and 6 do not require this additional join and only requires a basic mapping:

$NET(Q_1, Q_2, Q_3) = \pi_{A^{\text{local}}_{\text{title}}}(R^{\text{local}}) \longmapsto_{(1:1)} \pi_{A^{\text{join}}_{\text{title}}}(R^{\text{join}}_{\text{title}}) \wedge \pi_{A^{\text{local}}_{\text{year}}}(R^{\text{local}}) \longmapsto_{(1:1)} \pi_{A^{\text{join}}_{\text{production\_year}}}(R^{\text{join}}_{\text{title}}) \wedge \pi_{A^{\text{local}}_{\text{actor}}}(R^{\text{local}}) \longmapsto_{(1:1)} \pi_{A^{\text{join}}_{\text{name}}}(R^{\text{join}}_{\text{name}})$ where R^{join} is the projection of $= R^{\text{foreign}}_{\text{name}} \bowtie_{name.id=cast\_info.person\_id} R^{\text{foreign}}_{\text{cast\_info}} \bowtie_{cast\_info.movie\_id=title.id} R^{\text{foreign}}_{\text{title}}$.

However, when retrieving tuples that represent television series (2 and 4) the mapping is different and an additional join must be performed.

$NET(Q_1, Q_2, Q_3) = \pi_{A^{\text{local}}_{\text{title}}}(R^{\text{local}}) \longmapsto_{(1:1)} \pi_{A^{\text{join-tv}}_{\text{series\_title}}}(R^{\text{join-tv}}) \wedge \pi_{A^{\text{local}}_{\text{year}}}(R^{\text{local}}) \longmapsto_{(1:1)} \pi_{A^{\text{join-tv}}_{\text{production\_year}}}(R^{\text{join-tv}}) \wedge \pi_{A^{\text{local}}_{\text{actor}}}(R^{\text{local}}) \longmapsto_{(1:1)} \pi_{A^{\text{join-tv}}_{\text{name}}}(R^{\text{join-tv}})$ where the Name of the series must be recovered through an recursive join of the title relation $R^{\text{join-tv}} = R^{\text{join}} \bowtie_{title.episode\_of\_id=tv\_title.id} \rho_{title/tv\_title}(\rho_{title.title/title.series\_title}(R^{\text{foreign}}_{\text{title}}))$ that is in turn bound to the episode title $R^{\text{join}} = R^{\text{foreign}}_{\text{name}} \bowtie_{name.id=cast\_info.person\_id} R^{\text{foreign}}_{\text{cast\_info}} \bowtie_{cast\_info.movie\_id=title.id} R^{\text{foreign}}_{\text{title}}$.

In both these cases, the intermediate relation cast_info must be used to join the actor relation to the title relations. Table 6.8 shows the breakdown of the join similarity data for all queries. Overall the results are similar to that of the MythTV database, but with a lower discriminating power in the number of joins that must be attempted on the database before the correct one is found.

The generation of the translations that form part of the integration mappings was not found to be any more difficult than within the MythTV database in that few translations were required. Generally, the translations were a simple one-to-one copy of the attributes instance data. For a television series, the translations were:

$A^{\text{local}}_{\text{title}}\tau'^{\%'}_{(1:1)}A^{\text{join-tv}}_{\text{series\_title}}\ A^{\text{local}}_{\text{year}}\tau'^{\%'}_{(1:1)}A^{\text{join-tv}}_{\text{production\_year}}\ A^{\text{local}}_{\text{actor}}\tau'^{\%'}_{(1:1)}A^{\text{join-tv}}_{\text{name}}\ A^{\text{local}}_{\text{sub-title}}\tau'^{\%'}_{(1:1)}A^{\text{join-tv}}_{\text{title}}$

while for a movie the translation was:

$A^{\text{local}}_{\text{title}}\tau'^{\%'}_{(1:1)}A^{\text{join}}_{\text{title}}\ A^{\text{local}}_{\text{year}}\tau'^{\%'}_{(1:1)}A^{\text{join}}_{\text{production\_year}}\ A^{\text{local}}_{\text{actor}}\tau'^{\%'}_{(1:1)}A^{\text{join}}_{\text{name}}$

All of the other mappings were found through a similarity search as with the MythTV database, however the search for the genre data was particularly interesting. A fragment of the IMDB database is represented in Figure 6.6 where the genre information is encoded along with other assorted movie information.

In order to find the movie genre information, it is necessary to find the 'Science Fiction' string within the database. This was easily found as well as the required join through a key /

Query-Net	Number of suggested joins	correct join rank	Joins queried
1/1-3	13	7	6
1/2	9	1	
2/1-3	12	3	10
2/2	10	5	
3/1-3	11	3	10
3/2	10	5	
4/1-3	12	5	10
4/2	10	5	
5/1-3	13	2	11
5/2	11	5	
6/1-3	11	4	11
6/2	11	4	

Table 6.8: Comparison of key / foreign key location results for all queries.

foreign key relationship in the manner of $A^{\text{local}}_{\text{genre}}\tau'^{\%'}_{(1:1)}A^{\text{join}}_{\text{info}}$ with the appropriate join $R^{\text{join}} = R^{\text{join}} \bowtie_{movie\_info.movie\_id=title.id} R^{\text{foreign}}_{\text{movie\_info}}$.

However by inspection of Figure 6.6, it becomes obvious that not only will genre information be retrieved from the database but also distribution, soundtrack and copyright information, since the relation also contains information beyond genre. Within the database, this information is labelled using the info_type relation. In effect, to obtain only the information that we are interested in, the condition $\sigma_{\text{movie\_info=info\_type=3}}$ ($R^{\text{foreign}}_{\text{movie\_info}}$) would be required to ensure that the mapping would be accurate.

This problem lies in a design decision of the IMDB database where assorted information such as genre has been concentrated within the same attribute of a relation. This design decision is common when designers do not wish to incur the cost of additional attributes or relations for seldom used information. Furthermore, IMDB allows for a movie to have multiple genres, which would require a large relation to encode the $n : m$ relationship.

This same problem occurs for the mapping of the description attribute that also requires information from the movie_info attribute of the foreign database. However, in this case the condition that we would require would be $\sigma_{\text{movie\_info=info\_type=102}}$ instead.

Hence, in order for us to accurately retrieve the information from this relation, we again have to apply a condition to the retrieval operation. Unlike the conditions for the MythTV database, however, we do not have a simple test for a null value to discriminate the encodings. Perhaps we

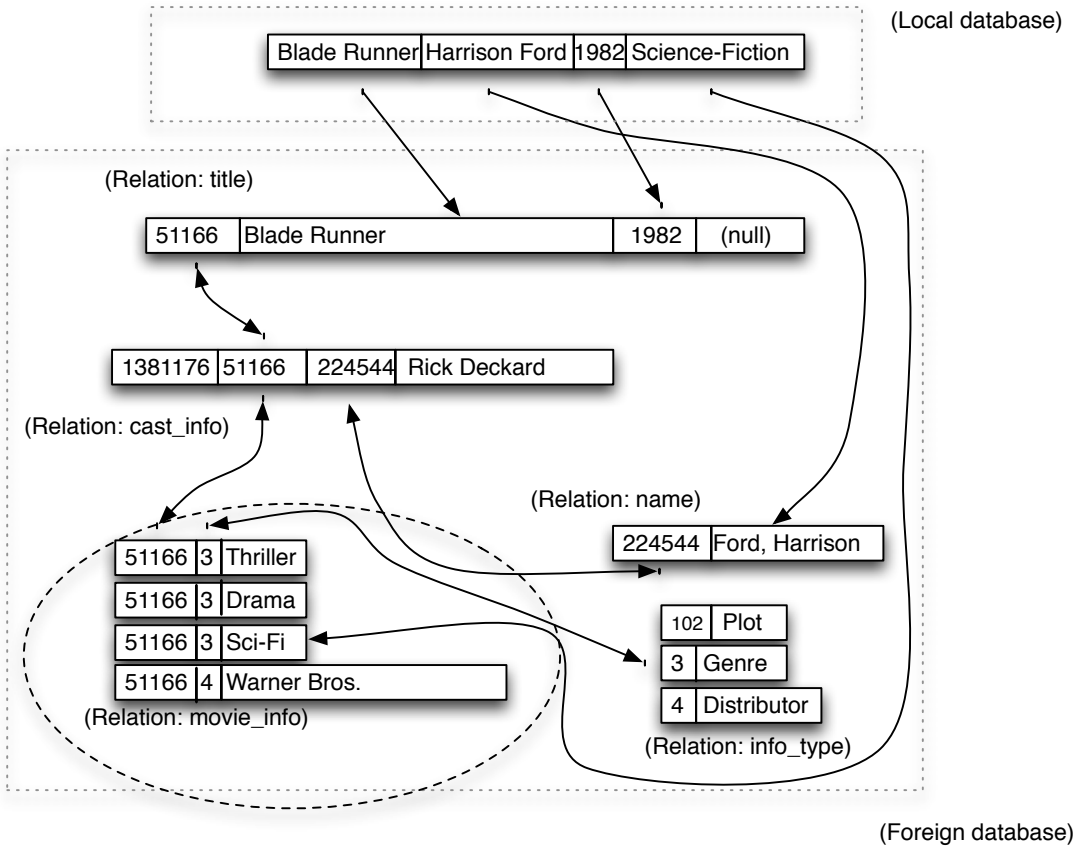

Figure 6.6: The genre information cannot be extracted because of the database design.

could use of a pivot operator to transform a join of the movie_info and info_type relations, in the manner proposed by Fletcher [38]. While this would not result in pure SQL query, we would then be able to acquire single attributes named Genre and Plot from the foreign database for mapping to our local database.

Another problem illustrated by the IMDB database is shown in Figure 6.7, where a specific case highlights some of the design decisions of the database and the behaviour of our targeted integration approach. To deal with localised and re-masters versions of movies, a special relation exists within the IMDB database that links any alternate media release to its original theatrical release.

We serendipitously avoided this situation with our query set because we queried the original theatrical and un-translated releases. When a query is attempted that makes use of this particular

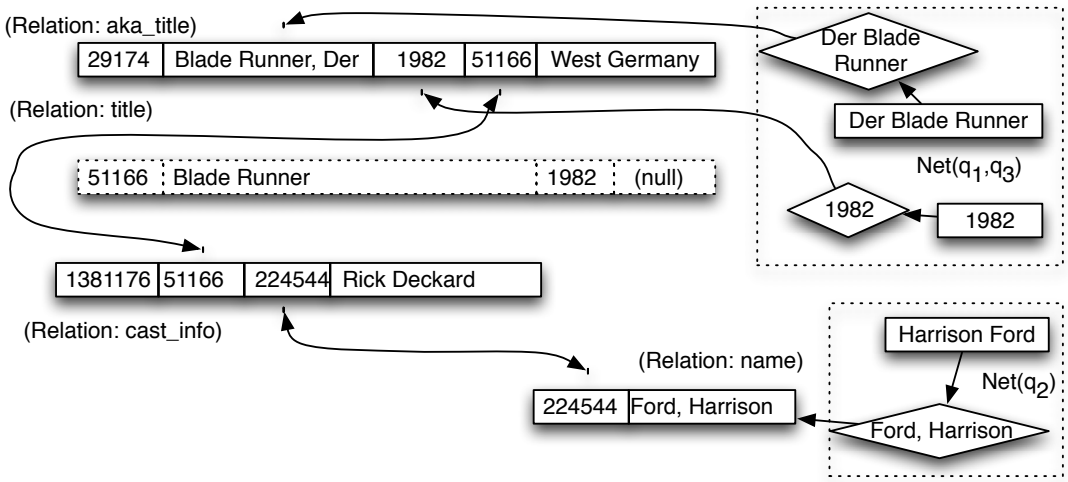

Figure 6.7: Alternate movie releases can also be accesses, but it is unclear how this would be correct

relation, the end result is usually failure as the mapping cannot support the rest of the local database. Interestingly, we were able to avoid this problem because of the targeted nature of the integration method.

These competing designs and internal mappings highlight the need to be able to use multiple integration solutions concurrently. In the earlier case of television show versus movies, both solutions map the $A^{\text{local}}_{\text{title}}$ attribute to the same attribute within the foreign database, but using two different join paths. For either type of show that the user query represents (e.g.: a movie or a TV series), there exists at least one local tuple that support the translations. Hence, the 'correct' translation is selected for the show type that the user chooses in his example query. If we omit any example tuples from the local database that match the user's show query type, the method will fail to find a transformation as no support can be found for the user's requirements.

As for the MythTV database, we can examine what an integration solution must include to handle both movies and television series from IMDB. In the case of a TV episode, the episode_of_id attribute of relation title must not be null in order for a join to occur which retrieves the TV series title. Conversely, a movie is not part of a TV series and hence the attribute episode_of_id will be null, which makes a join impossible. Since these attribute value constraints can be derived directly from the joins in a translation formula, we can easily check that the current record is appropriate for an individual translation formula.

The remaining concern is the question of whether the translation formulas overlap, or collide, in which case we are unable to use them concurrently. We can deduce this directly from the conditions of the translation formulas. In this case, the episode_of_id attribute cannot be null and non-null at the same time, hence we can deduce that both translation formulas can co-exist without verifying the foreign database itself.

In more complex cases where such logical deductions are not available, we can infer the appropriateness of multiple translations by searching the foreign database for a tuple that can match the conditions of more than one translation formula. For example, should our relations contain the fictitious attributes is_a_TV_series and is_a_movie, the presence of a tuple with both attributes marked as true would show both translations to be incompatible.

Query 6.3 presents the SQL query that would be used to retrieve the movie information from the IMDB database. Currently, we have no means of dealing with the design of the database for certain information such as plot description and movie genre. The query makes use of query constraints that we are currently unable to discover to map this information. We use a similar approach in Query 6.4 to extract television series from the database. The difference is that we add a recursive join to find the appropriate series title. Note that in both cases, we are able to differentiate between the two types of media by enforcing the requirement of one of the solutions that the episode_of_id attribute be available to perform the recursive join. Again, the question of how to mechanise this decision process of whether the solutions can function concurrently is something that we wish to explore in future work.

Query 6.3 SQL query mapping the movie contents of the IMDB foreign database.

```
select title.title as local.title,
title.production_year as local.airdate,
name.name as local.cast,
movie_info.info as local.genre,
second_movie_info.info as local.description,
kind_type.kind as local.type
from title, name, cast_info, movie_info, kind_type, movie_info as
second_movie_info,
where cast_info.person_id=name.id
and cast_info.movie_id=title.id
and movie_info.movie_id=title.id
and second_movie_info.movie_id=title.id
and kind_type.id=title.kind_id
```

Query 6.4 SQL query mapping the television series contents of the IMDB foreign database.

```
select series_title.title as local.title,
title.title as local.subtitle,
title.production_year as local.airdate,
name.name as local.cast,
movie_info.info as local.genre,
second_movie_info.info as local.description,
kind_type.kind as local.type
from title as series_title, title, name,
cast_info, movie_info, kind_type, movie_info as
second_movie_info,
where cast_info.person_id=name.id,
series_title.id=title.episode_of_id,
and cast_info.movie_id=title.id
and movie_info.movie_id=title.id
and second_movie_info.movie_id=title.id
and kind_type.id=title.kind_id
```

6.4 Discussion

One of the unexpected elements of this research was the ease with which joins could be located between relations and the difficulty with which some of their attribute relationships could be enumerated. In all cases, it was obvious from inspecting the top-k results manually which relation and attribute a query term would be linked too. However, it was found that deducing what attributes were actual key / foreign key relationships was difficult, in that spurious dependencies would appear against unused or constant valued attributes. In these tests, we were able to remove some of the superfluous join conditions through the use of the additional example tuples $\mathcal{T}^{\text{local}-\text{ex}}$.

One of the items that was revealed is that database designers tend to use integers as a means of 'flagging' tuples. For example, the hdtv attribute of the program relation is an integer that takes a 1 or 0 value depending if the show is in High Definition TV. Such integer flags tend to begin at 0 and begin counting up, which results in a disproportionate number of attributes with a few small integers only. These also tend to generate many spurious join conditions between themselves, which requires a pruning step to ensure that all join conditions are necessary. There may be some additional methods of normalisation that could be applied to key / foreign key searches to discourage the inclusion of these attributes when generating join conditions.

Previous work in the area of join finding reported that the generation of spurious large joins

over faulty joins was a problem. Because we restricted the tuples that are considered to locate joins with the initial query, no cases of large joins occurred. What we did find in several cases was that superfluous join conditions would be located that were unnecessary to the join and would prevent the sampled tuples from matching the found joins. Hence, a trimming operation was added for this specific case, as reviewed in Algorithm 9 of Chapter 4.

Hence, the identification of specific tuples that are shared by different queries is preferable to the simple search for join conditions over several tuples. This requires us to keep track of individual tuples instead of the attribute values themselves.

A difficulty specifically with the MythTV database is the amount of information that it contains which is duplicated, ambiguous or inactive. In many cases, the search for networks was slowed by attributes that had similar content to our search. For example, the database contains information on available video game titles. These are not related to the Movie and TV line-ups, but video game releases are usually themed on feature movies and named as such. While these values were returned in our initial retrieval of instances from the database, it was not logically possible to achieve a join path to other line-up information and the non-relevant video game information was ignored. This illustrates the robustness of our approach.

One relation containing old data is oldprogram, which contains obsolete line-up information. As some queries sent to the database matched the information within the old lineup, the initial networks contained the relation as a possible match. However, because the information was incomplete both in the number of tuples and attributes (the relation was a subset of the active attributes), these paths were soon dropped and the correct network was constructed using the active program relation.

We were also successful in targeting the subject of the database integration between both databases through the use of a query. The problem that arose was that we were too successful: we would only acquire either television series or movies from either databases due to their differences in encoding. This is interesting in that in many cases, a simple query is sufficient for the integration process to take place at the cost of a more generalised solution. A possibility is the use of multiple queries to allow for some diversity in the type of object being retrieved, but this demands some kind of higher integration logic capable of deciding on the precedence of multiple integration solutions.

Finally, one of the concerns is that databases do not always follow straightforward representations for the storage of their information. Because this method depends on the relation as its primary means of solving the integration problem, it can fail as in the case of the IMDB database for the genre and description attributes. In the future work we should explore means of locating

such situations and re-evaluating the data to fit our method.

There is also the question of how the method handles ambiguous queries that, for example, could refer to both a motion picture or a television series episode. While we were unable to find data or query cases where this occurs within either database, we can infer the behaviour of the method from its design.

Hypothetically, should we query the database with the terms "William Shatner" and "Star Trek", it would be ambiguous whether we are referring to a movie or a television series. Assuming that the attribute instances were to match both queries exactly, the returned integration solution would always be the shortest path one. Hence, in the case of the IMDB database the motion picture would be joined to the actor, because it would only require the relations $R^{\text{foreign}}_{\text{title}} \bowtie_{\text{title.id=cast\_info.movie\_id}} R^{\text{foreign}}_{\text{cast\_info}} \bowtie_{\text{name.id=cast\_info.person\_id}} R^{\text{foreign}}_{\text{name}}$. Conversely, retrieving the television series alternative for this query would imply the joins $\rho_{title/tv\_title}\ (\ R^{\text{foreign}}_{\text{title}}\ ) \bowtie_{\text{tv\_title.id=title.episode\_of\_id}} R^{\text{foreign}}_{\text{title}} \bowtie_{\text{title.id=cast\_info.movie\_id}} R^{\text{foreign}}_{\text{cast\_info}} \bowtie_{\text{name.id=cast\_info.person\_id}} R^{\text{foreign}}_{\text{name}}$. Because of the 'greedy' nature of the search algorithm, relation to relation joins are attempted before using intermediate relations, the solution with the least number of joins will be returned, in this case the movie.

In all cases, the overall method will find a relation that assembles as many of the terms within the query as possible, in the simplest way possible.

Another issue is with the complex translation of attributes that merge information that is contained within multiple attributes in another database.

A common representation within a database system is to have the name of a person in two attribute instead of one, therefore one attribute for the family name and one for the common name. To this end the name attribute of the MythTV database was split into two attributes: last and first.

This case is handled in the way that networks are searched where inexact matches are used for the network searching. The lack of a specific tuple to search for in a network does expand the number of network under consideration significantly.

Limiting the retrieval to 100 results per attribute search, there are 21 attributes over 13 relations that contain a possible in-exact match to 'Harrison Ford' within 1,031 retrieved tuples. 193 of these tuples could be capable of possibly forming a complete translation formula to the query term 'Harrison Ford', however only two of these have a potential key / foreign key relationship with the remaining query networks. Out of these two, both networks fail to support a translation formula which is complete.

Similarly, 741 of the retrieved instances could create a complete 1:n translation formula can

make one on the tuple, from which the same two query networks are extracted. As no network can be found with a matching translation, the translation is therefore found independently by retrieving additional instances from the database, using the example tuples over the relations that were previously retrieved by the query term. While hypothetically there could be many translations found concurrently, in this case only one is found, through the first and last attributes in Figure 6.8.

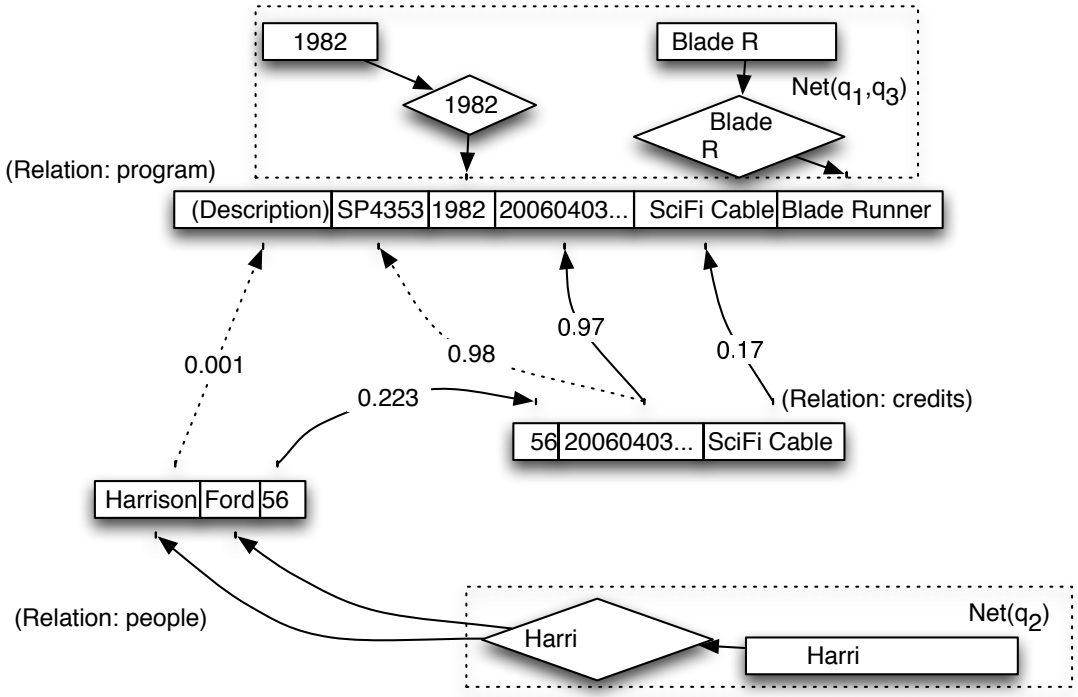

Figure 6.8: In this sub-case, the names of the actors are split across two attributes.

The located translation was $A^{\text{foreign}}_{\text{people}}\ \tau^{\text{first}[1-\text{inf}]+', '+\text{last}[1-\text{inf}]}_{2:1}\ A^{\text{local}}_{\text{cast}}$ using only the example tuples. With the translation providing a definite mapping and translation for the last query term, a linkage between the last two networks is searched for. This is done in the manner previously discussed: by iteratively searching every possible join path between the matching tuple sets of both networks. There exists only one case where the specific tuples can be joined, using the intermediate relation credit.

On of the elements that this case highlights is the necessity to attack mappings and translations in a manner that has the least cost. In the previous queries, such as the case of the MythTV

episode release dates, we would obtain a network and a translation very quickly because the number of tuples refereed to was small.

Here after attempting to find the translation and the network join paths concurrently and failing, we could have perused the translation by itself or the network join by itself. Since finding the network join is prohibitively expensive, we choose to locate the translation first at the cost of additional searches of a subset of the relations. This created a situation that is similar to the generalised case of translation finding and we were able to locate a correct mapping and translation for the attribute.

It is this mapping that allowed us to then locate the join path within the database. Without the translation the number of queries required to find the path would have been prohibitive.

6.5 Conclusion

While retrieving instances from the databases and creating initial networks, we had the same explosion in the number of potential networks as was reported by Agrawal et al. [4] and Mayssam et al. [101]. However, contrary to them we were able to rely on a few tuples from the local database to prune the networks to a small number, suitable for detailed exploration.

This reduction in the number of useful networks is similar to the results reported by Kotidis et al. [66], where in practice there are actually only one or two tuples that will satisfy the requirements of a query. However, Kotidis reported this result in the context of their tool being used by domain experts, supported by a large amount of meta-data for each database object. In our case, the reduction is an inherent part of an automated process that does not require interaction beyond the initial query and which does not use any meta-data.

Title	Sub-Title	Type	Genre	Year	Cast	Description
Dark Angel	Heat	Series	Fantasy	2000	Jessica Alba	...
Lethal Weapon	-	Movie	Action	1987	Danny Glover	...
MASH	Welcome to Korea	Series	Comedy	1975	Alan Alda	...
Puppets Who Kill	For Jesus	Series	Comedy	2004	Dan Redican	...
Queer Eye for the Straight Guy	Kord S	Series	Self impr.	2005	Ted Allen	...
Starship Troopers	-	Movie	Action	1997	Denise Richards	...
The Desert Rats	-	Movie	War	1953	James Mason	...
Wild Wild West	-	Movie	Action	1999	Will Smith	...
X2 X-Men United	-	Movie	Fantasy	2003	Hugh Jackman	...

Table 6.9: The tuples $\mathcal{T}^{\text{local}-\text{ex}}$, selected as local example tuples from the local database R^{local}.

Chapter 7

Conclusion

In this thesis we approached the problem of database integration by targeting a specific area of a foreign database for integration to a known, local database. A user defined, free-form query was used to define what information to integration to the local database. No schema or knowledge of the non-cooperating foreign database was assumed and no human intervention was involved in the searching, matching and translation process.

Fundamentally, the thesis is based on two observations: (1) the majority of the cost of processing queries on a database is due to number of tuples within a relation and (2) relations within relational databases bind items of information in a pattern that can be used to recognise entities. We list here the contributions of this thesis.

7.1 Summary of contributions

During the initial discovery phase of the database the previous state of the art was the use of a substring search or of a TF-IDF search (requiring the creation of an index). The proposed keyword searching solution achieved results with similar performance in terms of precision as that of TF-IDF retrieval, without requiring the creation of an index. Furthermore, it was found that because of the way that the index is created on the foreign database, the queries can be processed over four times faster than with a TF-IDF approach. In absolute terms, the complexity of the retrieval method is linear to the number of tuples within a relation.

Similarly, sampling has been used for years as a means of extracting only the minimum of information from a data set in order to analyse its contents. Whereas in previous work the

sampling of an attribute would require the traversal of the entire relation, we propose a method where a subset of the attribute is retrieved as part of the querying process and used as a directed sample of the attribute. This provides us with a method to obtain a sample of the relation comparable with a uniform sample of the attribute and sufficient to support attribute matching functions.

While the matching of attributes within and without databases has previously received attention, the problem of translation has been one that has seen very little work. We propose a generalised method for the discovery of translations between different database attribute in one-to-one and many-to-one translations. This method is self-directed and tolerant of poor attribute matching, to be novel. Its worst case computational complexity is $O(w * l * s^{\text{source}} * s^{\text{target}})$, where s^{source} is the number of tuples in the source attribute, s^{target} is the number of tuples in target aggregated attribute, l is the number of potential source attributes and w is the maximum number of characters in any value in the target aggregate attribute.

Finally, a method of finding appropriate key / foreign key relationships within databases is proposed. In previous methods the number of possible relationships was based on schema matching or data-driven approaches. In this approach we also use a data-driven approach, but restrict the possible tuples that may be used to match from one relation to another with a set of known example instances. While the theoretical computational complexity remains $n * m$ in the worst case, where both n and m refer to the number of matchable attributes within each relation, the actual runtime performance is much better. In an unconstrained case, any value match between any two attributes would be sufficient to suggest a key / foreign key relationship, and this results in a very large number of potential relationships found. In the new approach, the forcing of appropriate values on certain attributes within both relations constrains the key / foreign key finding to relations that match appropriate relationships between known attributes. The end result is a dramatic drop in the number of potential relations, from tens of thousands to a few dozen. The method is also extended to deal with key / foreign key relationships spanning multiple intermediate relations.

7.2 Future work

For convenience, we made the explicit assumption within the work that the local database D^{local} would be understood and its relation to the query terms $\mathcal{Q}$ known through a pre-determined map $\mathcal{MAP}()$.

It would be interesting to remove this assumption, so that both local and foreign databases

would be concurrently searched and their key / foreign key located. The individual relations within each database would provide a partial crib-sheet for locating key / foreign key relations within the other database as more complex relations are iteratively found. Should we be able to extend this method to process multiple foreign databases, we could then achieve a process where the local database D^{local} would be created out of multiple other databases without the need for local example tuples.

However, one of the major conclusions from Chapter 6 was the need to support multiple concurrent integration solutions. Similarly, our translation finding algorithm in Chapter 5 can only function with a single translation solution at a time. Because the encoding of information can take multiple representations within the same relations and key / foreign key joins, a method must be found to allow for the aggregation of multiple concurrent solutions. The problem lies in automating the analysis of the integration solutions to find the conditions under which they can co-exist and how to encode these conditions in a manner that the database can implement.

Finally, the thesis approaches integration with a directed method where only the necessary relations are used. This is done to prevent unwanted information from being imported. In some cases it may be that there are several other clusters of relations that require integration, but that have no key / foreign key relationships. In this case, a strategy of segmentation may be interesting: clusters of relations could be formed from several initial query searches so that only specific sets of relations would be searched for the remaining query terms.

Appendix A

List of thesis conventions and variables

As a generalisation, any variable typeset in a calligraphic style, e.g.: $\mathcal{S}$, represents a set of objects while normal type, e.g.: S, represents an object. Superscript refers to the specific database, or database subset, while indices refer to the specific attribute, tuple or relation with a database. Any star "$*$" character within an index indicates a don't care condition.

D^{local} refers to the local database, under the control of the integrating process.

R^{local} refers to the local relation which is a pre-defined projection of all data within D^{local}.

$\mathcal{A}^{\text{local}}$ represents the set of all attributes within the local relation R^{local}. $L^{\text{local}} = |A^{\text{local}}|$, where A_l^{local} refers to the lth attribute of relation R^{local}.

$\mathcal{Q}$ represents the set of query terms. $I = |\mathcal{Q}|$, while Q_i represents the ith query term.

$\mathcal{T}^{\text{local}-\text{ex}}$ the set of example tuples that have been selected from R^{local}. $\text{t}_{mi}^{\text{local}-\text{ex}}$ represents the mth instance matching ith terms of $\mathcal{Q}$. Only the 1st of M tuples is retrieved based on the terms Q_i. The remaining $M - 1$ tuples are retrieved from the local database.

$\mathcal{MAP}()$ is a mapping between any Q_i of $\mathcal{Q}$ and R^{local}.

D^{foreign} refers to the foreign database, which is uncontrolled by the integrating process.

$\mathcal{R}^{\text{foreign}}$ represents the set of relations within the foreign database. $J = |R^{\text{foreign}}|$ and R_j^{foreign} represents the jth relation within the foreign database.

$\mathcal{A}^{\text{foreign}}$ represents the set of all attributes within the foreign database, while $\mathcal{A}_j^{\text{foreign}}$ is the subset of all attributes that belong to the jth relation within the foreign database. $L_j^{\text{foreign}} = |A_{jl}^{\text{foreign}}|$ andA_{jl}^{foreign} refers to the lth attribute of relation R_j^{foreign}.

$\mathcal{T}_j^{\text{foreign}}$ refers to the complete set of tuples within relation j of the foreign database, while T_j^{foreign} refers to a specific tuple.

$\mathcal{I}_{jl}^{\text{foreign}}$ refers to the complete set of instances within relation j, attribute l of the foreign database, while I_{jl}^{foreign} refers to a specific value.

A "triplet" is the grouping of a specific relation R, attribute A and value t. Whenever we wish to reference a specific location within the database, we use this term to prevent confusion.

$\mathcal{I}_{jli}^{\text{foreign\_match}}$ is the ranked set of instances that were retrieved from attribute A_{jl}^{foreign} based on query term Q_i. $I_{jlie}^{\text{foreign\_match}}$ indicates the eth ranked instance out of E_{jli} retrieved.

$\mathcal{T}_{jli}^{\text{foreign\_match}}$ specifies the set of tuples retrieved when querying attribute A_{jl}^{foreign} based on query term Q_i. $T_{jlie}^{\text{foreign\_match}}$ refers to the eth tuple ranked according to $I_{jlie}^{\text{foreign\_match}}$.

Note that we can address a specific attribute within $T_{jlie}^{\text{foreign\_match}}$ by adding the subscript p to get $I_{jliep}^{\text{foreign\_match}}$. Therefore if $p = l$, then $I_{jliep}^{\text{foreign\_match}} = I_{jlie}^{\text{foreign\_match}}$.

$\mathcal{T}_{ie}^{\text{query}}$ is the set of all instances that were retrieved for query term Q_i.

$\mathcal{T}_{jl}^{\text{attrib}}$ is the set of all instances that were retrieved from relation j, attribute l for all query terms $\mathcal{Q}$.

$\text{sim}(\mathcal{I}_1, \mathcal{I}_2)$ represents the similarity scores between a set of instance values $\mathcal{I}_1$ and $\mathcal{I}_2$.

$\mathcal{NET}()$ is a set of possible networks.

PLAN() Represents an integration plan that can be used by a mediator.

$A^{local} \longmapsto_{n:m} A^{foreign}$ Represents a certain mapping from A^{local} to $A^{foreign}$. n:m represents the cardinality of the mapping, where we have information from n different A^{local}'s mapping to m different $A^{foreign}$'s.

$A^{local} \tau_{n:m}^{[x-y]} A^{foreign}$ Represents a certain translation from A^{local} to $A^{foreign}$. n:m represents the cardinality of the translation, where we have n A^{local}'s and m $A^{foreign}$. $[x-y]$ represents the editing formula from the source to the target. Note that $A^{local} \tau_{n:m}^{[x-y]} A^{foreign}$ implies

$A^{local} \longmapsto_{n:m} A^{foreign}$, but not the converse as τ contains translation information that $\longmapsto$ does not have.

$\mathrm{R}^{\mathrm{join}}$ The relation formed by joining all of the relations pointed to by NET().

$\sigma_{R^{\mathrm{foreign}} \forall \tau_{(n:1)} R_x^{\mathrm{local-ex}}}$ To enhance the readability of certain complex querying operations, we use $\forall$ as a short hand meaning for all available mappings and / or translations with the current plan between R^{foreign} and $R_x^{\mathrm{local-ex}}$. Here the expression would read 'apply all $n:1$ translations that translate any element of R^{foreign} to $R_x^{\mathrm{local-ex}}$'.

Bibliography

[1] S. Adali and V. Subrahmanian. Intelligent caching in heterogeneous reasoning and mediator systems. In N. Mars, editor, *Proc. of the Second International Conference on Building and Sharing of Very Large-Scale Knowledge Bases*, pages 247–256. IOS Press, May 1995.

[2] Douglas Adams. *The Hitchhiker's Guide to the Galaxy.* Pan Books, 1979.

[3] B. Aditya, Gaurav Bhalotia, Soumen Chakrabarti, Arvind Hulgeri, Charuta Nakhe, Parag, and S. Sudarshan. BANKS: Browsing and keyword searching in relational databases. In *VLDB 2002, Proceedings of 28th International Conference on Very Large Data Bases*, pages 1083–1086. Morgan Kaufmann, August 2002.

[4] Sanjay Agrawal, Surajit Chaudhuri, and Gautam Das. DBXplorer: A system for keyword-based search over relational databases. *icde*, 00:0005, 2002.

[5] Davide Alberani. IMDB converter script, October 2006.

[6] Noga Alon, Phillip B. Gibbons, Yossi Matias, and Mario Szegedy. Tracking join and self-join sizes in limited storage. In *PODS '99: Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 10–20, New York, NY, USA, 1999. ACM Press.

[7] Yigal Arens, Chun-Nan Hsu, and Craig A. Knoblock. Query processing in the SIMS information mediator. In Michael N. Huhns and Munindar P. Singh, editors, *Readings in Agents*, pages 82–90. Morgan Kaufmann, San Francisco, CA, USA, 1997.

[8] David Aumueller, Hong Hai Do, Sabine Massmann, and Erhard Rahm. Schema and ontology matching with COMA++. In Fatma Özcan, editor, *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 906–908. ACM, 2005.

[9] Domenico Beneventano, Sonia Bergamaschi, Silvana Castano, Alberto Corni, R. Guidetti, G. Malvezzi, Michele Melchiori, and Maurizio Vincini. Information integration: The MOMIS project demonstration. In Amr El Abbadi, Michael L. Brodie, and et al., editors, *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases*, pages 611–614. Morgan Kaufmann, 2000.

[10] S. Benkley, J. Fandozzi, E. Housman, and G. Woodhouse. Data element tool-based analysis (DELTA). Technical Report MTR-95B147, MITRE Corp., 1995.

[11] S. Bergamaschi, S. Castano, and M. Vincini. Semantic integration of semistructured and structured data sources. *SIGMOD Record*, 28(1):54–59, 1999.

[12] Jacob Berlin and Amihai Motro. Autoplex: Automated discovery of content for virtual databases. In *CooplS '01: Proceedings of the 9th International Conference on Cooperative Information Systems*, pages 108–122, London, UK, 2001. Springer-Verlag.

[13] Jacob Berlin and Amihai Motro. Database schema matching using machine learning with feature selection. In Anne Banks Pidduck, John Mylopoulos, Carson C. Woo, and M. Tamer Özsu, editors, *CAiSE*, volume 2348 of *Lecture Notes in Computer Science*, pages 452–466. Springer, 2002.

[14] Philip Bohannon, Eiman Elnahrawy, Wenfei Fan, and Michael Flaster. Putting context into schema matching. In *VLDB 2006: Proceedings of the 32nd International Conference on Very Large Data Bases*, pages 307–318, 2006.

[15] S. Bressan, Cheng Hian Goh, T. Lee, S. Madnick, and M. Siegel. A procedure for mediation of queries to sources in disparate contexts. In *ILPS '97: Proceedings of the 1997 international symposium on Logic programming*, pages 213–227, Cambridge, MA, USA, 1997. MIT Press.

[16] Sergey Brin. Extracting patterns and relations from the world wide web. In *WebDB '98: Selected papers from the International Workshop on The World Wide Web and Databases*, pages 172–183, London, UK, 1999. Springer-Verlag.

[17] Nicolas Bruno, Surajit Chaudhuri, and Luis Gravano. Top-k selection queries over relational databases: Mapping strategies and performance evaluation. *ACM Transaction on Database Systems*, 27(2):153–187, 2002.

[18] J. Bunge and M. Fitzpatrick. Estimating the number of species: A review. *Journal of the American Statistical Association*, 88(421):364–373, March 1993.

[19] Michael J. Carey, Laura M. Haas, Peter M. Schwarz, Manish Arya, William F. Cody, Ronald Fagin, Myron Flickner, Allen Luniewski, Wayne Niblack, Dragutin Petkovic, Joachim Thomas II, John H. Williams, and Edward L. Wimmers. Towards heterogeneous multimedia information systems: The GARLIC approach. In *RIDE-DOM*, pages 124–131, 1995.

[20] Paulo Carreira and Helena Galhardas. Execution of data mappers. In *IQIS '04: Proceedings of the 2004 international workshop on Information quality in information systems*, pages 2–9, New York, NY, USA, 2004. ACM Press.

[21] S. Castano and A. Ferrara. Knowledge representation and transformation in ontology-based data integration. In *Proc. of ECAI Workshop on Knowledge Transformation for the Semantic Web*, Lyon, France, July 2002.

[22] Moses Charikar, Surajit Chaudhuri, Rajeev Motwani, and Vivek Narasayya. Towards estimation error guarantees for distinct values. In *PODS '00: Proceedings of the nineteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 268–279, New York, NY, USA, 2000. ACM Press.

[23] Surajit Chaudhuri, Kris Ganjam, Venkatesh Ganti, and Rajeev Motwani. Robust and efficient fuzzy match for online data cleaning. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 313–324, 2003.

[24] Surajit Chaudhuri, Rajeev Motwani, and Vivek Narasayya. Histogram construction using adaptive random sampling with cross-validation for database systems. Technical Report 6278989, U.S. Patent Office, Redmond, WA, August 1998.

[25] Surajit Chaudhuri, Rajeev Motwani, and Vivek Narasayya. Random sampling for histogram construction: how much is enough? In *SIGMOD '98: Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 436–447, New York, NY, USA, 1998. ACM Press.

[26] Chris Clifton, E. Housman, and Arnon Rosenthal. Experience with a combined approach to attribute-matching across heterogeneous databases. In *DS-7*, pages 428–, 1997.

[27] Gordon V. Cormack and Thomas R. Lynam. Statistical precision of information retrieval evaluation. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 533–540, New York, NY, USA, 2006. ACM.

[28] Tamraparni Dasu, Theodore Johnson, S. Muthukrishnan, and Vladislav Shkapenyuk. Mining database structure; or, how to build a data quality browser. In Michael J. Franklin, Bongki Moon, and Anastassia Ailamaki, editors, *SIGMOD '02: Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 240–251. ACM, 2002.

[29] The Internet Movie Database. Alternate interfaces, November 2006.

[30] Robin Dhamankar, Yoonkyong Lee, AnHai Doan, Alon Halevy, and Pedro Domingos. iMAP: discovering complex semantic matches between database schemas. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 383–394. ACM Press, 2004.

[31] AnHai Doan, Pedro Domingos, and Alon Y. Halevy. Reconciling schemas of disparate data sources: a machine-learning approach. In *SIGMOD '01: Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, pages 509–520, New York, NY, USA, 2001. ACM Press.

[32] David W. Embley, Li Xu, and Yihong Ding. Automatic direct and indirect schema mapping: experiences and lessons learned. *SIGMOD Record*, 33(4):14–19, 2004.

[33] R. Fagin, P. Kolaitis, R. Miller, and L. Popa. Data exchange: Semantics and query answering. In *ICDT 2003, Siena, Italy*, 2003.

[34] Ronald Fagin, Phokion G. Kolaitis, and Lucian Popa. Data exchange: getting to the core. *ACM Transaction on Database Systems*, 30(1):174–210, 2005.

[35] C. T. Fan, Mervin E. Muller, and Ivan Rezucha. Development of sampling plans by using sequential (item by item) selection techniques and digital computers. *Journal of the American Statistical Association*, 57(298):387–402, June 1962.

[36] I. P. Fellegi and A. B. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, (64):1183–1210, 1969.

[37] R. A. Fisher, A. S. Corbet, and C. B. Williams. The relation between the number of species and the number of individuals in a random sample of an animal population. *Journal of Animal Ecology*, (12):42–58, 1943.

[38] George H. L. Fletcher. The data mapping problem: Algorithmic and logical characterizations. In *Workshop on Databases For Next Generation Researchers at ICDE*, 2005.

[39] George H. L. Fletcher. The data mapping problem: Algorithmic and logical characterizations. In *International Special Workshop on Databases For Next Generation Researchers (SWOD) at ICDE 2005*, 2005.

[40] E. Franconi, G. Kuper, A. Lopatenko, and I. Zaihrayeu. The CODB robust peer-to-peer database system. In *Proceedings of the 2nd Workshop on Semantics in Peer-to-Peer and Grid Computing (SemPGrid'04)*, 2004.

[41] Ariel Fuxman, Phokion G. Kolaitis, Renée J. Miller, and Wang-Chiew Tan. Peer data exchange. *ACM Transaction on Database Systems*, 31(4):1454–1498, 2006.

[42] Sumit Ganguly, Phillip B. Gibbons, Yossi Matias, and Avi Silberschatz. Bifocal sampling for skew-resistant join size estimation. In *SIGMOD '96: Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, pages 271–281, New York, NY, USA, 1996. ACM Press.

[43] Xianping Ge, Wanda Pratt, and Padhraic Smyth. Discovering chinese words from unsegmented text (poster abstract). In *Research and Development in Information Retrieval*, pages 271–272, 1999.

[44] Michael R. Genesereth, Arthur M. Keller, and Oliver M. Duschka. Infomaster: An information integration system. In Joan Peckham, editor, *SIGMOD '97: Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, pages 539–542. ACM Press, 1997.

[45] Lise Getoor, Nir Friedman, Daphne Koller, Avi Pfeffer, and Benjamin Taskar. Probabilistic relational models. In L. Getoor and B. Taskar, editors, *An Introduction to Statistical Relational Learning*. MIT Press, 2007.

[46] Phillip B. Gibbons and Yossi Matias. New sampling-based summary statistics for improving approximate query answers. In *SIGMOD '98: Proceedings of the 1998 ACM SIGMOD*

international conference on Management of data, pages 331–342, New York, NY, USA, 1998. ACM Press.

[47] Cheng Hian Goh, Stéphane Bressan, Stuart Madnick, and Michael Siegel. Context interchange: new features and formalisms for the intelligent integration of information. *ACM Transactions on Information Systems*, 17(3):270–270, 1999.

[48] Roy Goldman, Narayanan Shivakumar, Suresh Venkatasubramanian, and Hector Garcia-Molina. Proximity search in databases. In Ashish Gupta, Oded Shmueli, and Jennifer Widom, editors, *VLDB 1998, Proceedings of 24rd International Conference on Very Large Data Bases*, pages 26–37. Morgan Kaufmann, 1998.

[49] John Goldsmith. Unsupervised learning of the morphology of a natural language. *Computational Linguistics*, 27(2):153–198, 2001.

[50] I. J. Good and G. H. Toulmin. The number of new species, and the increase in population coverage, when a sample is increased. *Biometrika*, (43):45–63, 1956.

[51] L. Gravano, P. Ipeirotis, N. Koudas, and D. Srivastava. Text joins in an rdbms for web data integration. In *Intl. WWW Conference*, pages 90–101, 2003.

[52] Peter J. Haas, Jeffrey F. Naughton, S. Seshadri, and Lynne Stokes. Sampling-based estimation of the number of distinct values of an attribute. In *VLDB '95: Proceedings of the 21th International Conference on Very Large Data Bases*, pages 311–322, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[53] D. S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Cummunications of the ACM*, 18(6):341–343, 1975.

[54] The MythTV homebrew PVR Project, May 2006. Schema version 0.19, `http://www.mythtv.org/`.

[55] Vagelis Hristidis, Luis Gravano, and Yannis Papakonstantinou. Efficient IR-Style keyword search over relational databases. In *VLDB 2003, Proceedings of 29th International Conference on Very Large Data Bases*, pages 850–861, 2003.

[56] Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. TANE: an efficient algorithm for discovering functional and approximate dependencies. *The Computer Journal*, 42(2):100–111, 1999.

[57] J. W. Hunt and T. G. Szymanski. A fast algorithm for computing longest common subsequences. *Cummunications of the ACM*, 20(5):350–353, 1977.

[58] International Organization for Standardization. *ISO/IEC 9075-1:1999: Information technology — Database languages — SQL — Part 1: Framework (SQL/Framework)*. 1999.

[59] Theodore Johnson, S. Muthukrishnan, and Irina Rozenbaum. Sampling algorithms in a stream operator. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 1–12, New York, NY, USA, 2005. ACM Press.

[60] T. G. Jones. A note on sampling a tape-file. *Communications of the ACM*, 5(6):343, 1962.

[61] Benny Kimelfeld and Yehoshua Sagiv. Efficient engines for keyword proximity search. In AnHai Doan, Frank Neven, Robert McCann, and Geert Jan Bex, editors, *WebDB*, pages 67–72, 2005.

[62] Benny Kimelfeld and Yehoshua Sagiv. Finding and approximating top-k answers in keyword proximity search. In Stijn Vansummeren, editor, *PODS*, pages 173–182. ACM, 2006.

[63] T. Kirk, A. Y. Levy, Y. Sagiv, and D. Srivastava. The information manifold. In C. Knoblock and A. Levy, editors, *AAAI Spring Symposium on Information Gathering in Distributed Heterogeneous Environments*, Stanford University, Stanford, California, 1995.

[64] Craig A. Knoblock, Steven Minton, Jose Luis Ambite, Naveen Ashish, Ion Muslea, Andrew Philpot, and Sheila Tejada. The ariadne approach to web-based information integration. *International Journal of Cooperative Information Systems*, 10(1-2):145–169, 2001.

[65] Serhiy Kosinov. Evaluation of n-grams conflation approach in text-based information retrieval. In *String Processing and Information Retrieval*, pages 136–142, 2001.

[66] Yannis Kotidis, Amélie Marian, and Divesh Srivastava. Circumventing data quality problems using multiple join paths. In *CleanDB*, 2006.

[67] Nick Koudas, Amit Marathe, and Divesh Srivastava. Flexible string matching against large databases in practice. In Mario A. Nascimento, M. Tamer Özsu, Donald Kossmann, Renée J. Miller, José A. Blakeley, and K. Bernhard Schiefer, editors, *VLDB 2004: Proceedings of the 30th International Conference on Very Large Data Bases*, pages 1078–1086. Morgan Kaufmann, 2004.

[68] T. A. Landers and Ronni Rosenberg. An overview of MULTIBASE. In *DDB*, pages 153–184, 1982.

[69] Maurizio Lenzerini. Data integration: a theoretical perspective. In *PODS '02: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 233–246, New York, NY, USA, 2002. ACM Press.

[70] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics - Doklady*, 10(8):707–710, Feb. 1966.

[71] Alon Y. Levy, Alberto O. Mendelzon, Yehoshua Sagiv, and Divesh Srivastava. Answering queries using views. In *Proceedings of the 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 95–104, San Jose, Calif., 1995.

[72] Wen-Syan Li and Chris Clifton. SEMINT: a tool for identifying attribute correspondences in heterogeneous databases using neural networks. *Data Knowl. Eng.*, 33(1):49–84, 2000.

[73] Witold Litwin. An overview of the multidatabase system MRDSM. In *ACM '85: Proceedings of the 1985 ACM annual conference on The range of computing : mid-80's perspective*, pages 524–533, New York, NY, USA, 1985. ACM Press.

[74] Fang Liu, Clement T. Yu, Weiyi Meng, and Abdur Chowdhury. Effective keyword search in relational databases. In Surajit Chaudhuri, Vagelis Hristidis, and Neoklis Polyzotis, editors, *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 563–574. ACM, 2006.

[75] Jayant Madhavan, Philip A. Bernstein, and Erhard Rahm. Generic schema matching with Cupid. In *VLDB 2001: Proceedings of the 27th International Conference on Very Large Data Bases*, pages 49–58. Morgan Kaufmann Publishers Inc., 2001.

[76] Ioana Manolescu, Daniela Florescu, and Donald Kossmann. Answering xml queries on heterogeneous data sources. In *VLDB '01: Proceedings of the 27th International Conference on Very Large Data Bases*, pages 241–250, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[77] Arturas Mazeika, Michael H. Böhlen, Nick Koudas, and Divesh Srivastava. Estimating the selectivity of approximate string queries. *ACM Transaction on Database Systems*, 32(2):12, 2007.

[78] Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *ICDE 2002, Proceedings of the 8th International Conference on Data Engineering*, pages 117–128. IEEE Computer Society, 2002.

[79] George A. Miller. WordNet: a lexical database for English. *Communications of the ACM*, 38(11):39–41, 1995.

[80] Renee J. Miller, Mauricio Hernandez, Laura M. Haas, Ling-Ling Yan, C. T. Howard Ho, Ronald Fagin, and Lucian Popa. The Clio project: Managing heterogeneity. *SIGMOD Record*, 30(1):78–83, 2001.

[81] Tova Milo and Sagit Zohar. Using schema matching to simplify heterogeneous data translation. In Ashish Gupta, Oded Shmueli, and Jennifer Widom, editors, *VLDB 1998, Proceedings of 24rd International Conference on Very Large Data Bases*, pages 122–133. Morgan Kaufmann, 1998.

[82] P. Mitra, G. Wiederhold, and J. Jannink. Semi-automatic integration of knowledge sources. In *Proc. of the 2nd Int. Conf. On Information FUSION'99*, 1999.

[83] Jeffrey F. Naughton and S. Seshadri. On estimating the size of projections. In *ICDT '90: Proceedings of the third international conference on database theory on Database theory*, pages 499–513, New York, NY, USA, 1990. Springer-Verlag New York, Inc.

[84] H. B . Newcombe, J. M. Kennedy, S. J. Axford, and A. P. JAMES. Automatic linkage of vital records. *Science*, 130:954–959, October 1959.

[85] F. Olken and D. Rotem. Random sampling from databases - a survey. In *Statistics & Computing*, volume 5, pages 25–42, March 1995.

[86] Frank Olken. *Random Sampling from Databases.* PhD thesis, University of California, Berkeley, Mailstop 50B-3238, 1 Cyclotron Road, Berkeley, California 94720, U.S.A., 1993.

[87] Locale registry procedures, Version 2. Technical Report G502, The Open Group, May 1995.

[88] Luigi Palopoli, Giorgio Terracina, and Domenico Ursino. Dike: a system supporting the semi-automatic construction of cooperative information systems from heterogeneous databases. *Softw. Pract. Exper.*, 33(9):847–884, 2003.

[89] Byung-Hoon Park, George Ostrouchov, Nagiza F. Samatova, and Al Geist. Reservoir-based random sampling with replacement from data stream. In Michael W. Berry, Umeshwar Dayal, Chandrika Kamath, and David B. Skillicorn, editors, *SDM*. SIAM, 2004.

[90] Michael S. Paterson and Vlado Dancik. Longest common subsequences. In *Math. Foundations of Comp. Sci.*, pages 127–142, 1994.

[91] Fuchun Peng and Dale Schuurmans. Self-supervised Chinese word segmentation. *Lecture Notes in Computer Science*, 2189:238+, 2001.

[92] L. Popa, Y. Velegrakis, M. Hernandez, R. J. Miller, and R. Fagin. Translating web data. In *28th International Conference for Very Large Databases (VLDB 2002)*, August 2002.

[93] Lucian Popa, Yannis Velegrakis, Renee J. Miller, Mauricio Hernandez, and Ronald Fagin. Translating web data. Technical Report CSRI 441, University of Toronto, 2002.

[94] Sriram Raghavan and Hector Garcia-Molina. Crawling the hidden web. In *Proceedings of the Twenty-seventh International Conference on Very Large Databases*, 2001.

[95] Erhand Rahm and Philip Bernstein. On matching schemas automatically. Technical Report MSR-TR-2001-17, Microsoft Research, Redmond, WA, February 2001.

[96] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10(4):334–350, 2001.

[97] Milton Rokeach. *The Three Christs of Ypsilanti: A Psychological Study*. Random House Trade Paperbacks, January 1964.

[98] Neil C. Rowe. Antisampling for estimation: an overview. *IEEE Trans. Softw. Eng.*, 11(10):1081–1091, 1985.

[99] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA, 1986.

[100] Gerard Salton, A. Wong, and C. S. Yang. A Vector Space Model for Automatic Indexing. *Cummunications of the ACM*, 18(11):613, 1975.

[101] Mayssam Sayyadian, Hieu LeKhac, AnHai Doan, and Luis Gravano. Efficient keyword search across heterogeneous relational databases. In *ICDE 2007, Proceedings of the 13th International Conference on Data Engineering*. IEEE Computer Society, 2007.

[102] Len Seligman, Arnon Rosenthal, Paul Lehner, and Angela Smith. Data integration: Where does the time go? Technical report, MITRE Corp., Nov. 2005. MITRE Corporation.

[103] Amit P. Sheth, Sanjeev Thacker, and Shuchi Patel. Complex relationships and knowledge discovery support in the InfoQuilt system. *VLDB Journal*, 12(1):2–27, 2003.

[104] Luo Si and Jamie Callan. Using sampled data and regression to merge search engine results. In *SIGIR '02: Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 19–26, New York, NY, USA, 2002. ACM Press.

[105] Divesh Srivastava and Yannis Velegrakis. Intensional associations between data and metadata. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 401–412, New York, NY, USA, 2007. ACM.

[106] Marjorie Templeton, David Brill, Arbee L. P. Chen, Son Dao, and Eric Lund. Mermaid - experiences with network operation. In *ICDE 1986, Proceedings of the Second International Conference on Data Engineering*, pages 292–300, 1986.

[107] Leo Tolstoy. *War and Peace*. Project Gutenberg, 2001.

[108] Esko Ukkonen. Approximate string-matching with q-grams and maximal matches. *Theor. Comput. Sci.*, 92(1):191–211, 1992.

[109] V. Ventrone and S. Heiler. Some practical advice for dealing with semantic heterogeneity in federated database systems. In *Proceedings of the database colloquium*, august 1994.

[110] Jeffrey Scott Vitter. Faster methods for random sampling. *Communications of the ACM*, 27(7):703–718, 1984.

[111] Jeffrey Scott Vitter. An efficient algorithm for sequential random sampling. *ACM Trans. Math. Softw.*, 13(1):58–67, 1987.

[112] Robert H. Warren and Frank Wm. Tompa. Multi-column substring matching for database schema translation. In *Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB)*, Seoul, Korea, September 2006. ACM.

[113] William E. Winkler. Advanced methods for record linkage. Technical Report rr945, Statistical Research Division, U.S. Bureau of the Census., 1994.

[114] Misha Wolf and Charles Wicksteed. Date and time formats. Technical Report NOTE-datetime, W3 Consortium, September 1997.

[115] L. Xu and D. Embley. Discovering direct and indirect matches for schema elements. In *Proceedings of the 8th International Conference on Database Systems for Advanced Applications (DASFAA 2003)*, 2003.

[116] Ling Ling Yan, Renee J. Miller, Laura M. Haas, and Ronald Fagin. Data-driven understanding and refinement of schema mappings. In *SIGMOD '01: Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, pages 485–496. ACM Press, 2001.

[117] William Anthony Young. *Communication cost modeling for federated database systems.* Masters of mathematics, University of Waterloo, Waterloo, Ontario, Canada, 2005.