2010-12-16

Information Integration in a Grid Environment Applications in the Bioinformatics Domain

Ahmed M. Radwan
University of Miami, a.radwan@umiami.edu

UNIVERSITY OF MIAMI

INFORMATION INTEGRATION IN A GRID ENVIRONMENT
APPLICATIONS IN THE BIOINFORMATICS DOMAIN

By

Ahmed M. Radwan

A DISSERTATION

Submitted to the Faculty
of the University of Miami
in partial fulfillment of the requirements for
the degree of Doctor of Philosophy

Coral Gables, Florida

December 2010

UNIVERSITY OF MIAMI

A dissertation submitted in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy

INFORMATION INTEGRATION IN A GRID ENVIRONMENT
APPLICATIONS IN THE BIOINFORMATICS DOMAIN

Ahmed M. Radwan

Approved:

Akmal Younis, Ph.D. Associate Professor of Electrical and Computer Engineering	Terri A. Scandura, Ph.D. Dean of the Graduate School
Miroslav Kubat, Ph.D. Associate Professor of Electrical and Computer Engineering	Mei-Ling Shyu, Ph.D. Associate Professor of Electrical and Computer Engineering
Nigel John, Ph.D. Lecturer of Electrical and Computer Engineering	Sawsan Khuri, Ph.D. Assistant Research Professor of Human Genetics

RADWAN, AHMED M. (Ph.D., Electrical and Computer Engineering)
Information Integration in a Grid Environment *Applications in the Bioinformatics Domain* (December 2010)

Abstract of a dissertation at the University of Miami.

Dissertation supervised by Professor Akmal Younis
Number of Pages in Text. (162)

Grid computing emerged as a framework for supporting complex operations over large datasets, it enables the harnessing of large numbers of processors working in parallel to solve computing problems that typically spread across various domains. We focus on the problems of data management in a grid/cloud environment.

The broader context of designing a services oriented architecture (SOA) for information integration is studied, identifying the main components for realizing this architecture. The BioFederator is a web services-based data federation architecture for bioinformatics applications. Based on collaborations with bioinformatics researchers, several domain-specific data federation challenges and needs are identified. The BioFederator addresses such challenges and provides an architecture that incorporates a series of utility services; these address issues like automatic workflow composition, domain semantics, and the distributed nature of the data. The design also incorporates a series of data-oriented services that facilitate the actual integration of data. Schema integration is a core problem in the BioFederator context. Previous methods for schema integration rely on the exploration, implicit or explicit, of the multiple design choices that are possible for the integrated schema. Such exploration relies heavily on user interaction; thus, it is time consuming and labor intensive. Furthermore, previous methods have ignored the additional information that typically results from the schema matching process, that is, the weights and in some cases the directions that are associated with the correspondences. We propose a more automatic approach to schema integration that is based on the use of directed and weighted correspondences

between the concepts that appear in the source schemas. A key component of our approach is a ranking mechanism for the automatic generation of the best candidate schemas. The algorithm gives more weight to schemas that combine the concepts with higher similarity or coverage. Thus, the algorithm makes certain decisions that otherwise would likely be taken by a human expert. We show that the algorithm runs in polynomial time and moreover has good performance in practice. The proposed methods and algorithms are compared to the state of the art approaches. The BioFederator design, services, and usage scenarios are discussed. We demonstrate how our architecture can be leveraged on real-world bioinformatics applications. We preformed a whole human genome annotation for nucleosome exclusion regions. The resulting annotations were studied and correlated with tissue specificity, gene density and other important gene regulation features.

We also study data processing models on grid environments. MapReduce is one popular parallel programming model that is proven to scale. However, using the low-level MapReduce for general data processing tasks poses the problem of developing, maintaining and reusing custom low-level user code. Several frameworks have emerged to address this problem; these frameworks share a top-down approach, where a high-level language is used to describe the problem semantics, and the framework takes care of translating this problem description into the MapReduce constructs. We highlight several issues in the existing approaches and alternatively propose a novel refined MapReduce model that addresses the maintainability and reusability issues, without sacrificing the low-level controllability offered by directly writing MapReduce code. We present MapReduce-LEGOS (MR-LEGOS), an explicit model for composing MapReduce constructs from simpler components, namely, "Maplets", "Reducelets" and optionally "Combinelets". Maplets and Reducelets are standard MapReduce constructs that can be composed to define aggregated constructs describing the problem semantics. This composition can be viewed as defining a micro-workflow inside the MapReduce job. Using the proposed model, complex problem semantics can be defined in the encompassing micro-workflow provided by MR-LEGOS

while keeping the building blocks simple. We discuss the design details, its main features and usage scenarios. Through experimental evaluation, we show that the proposed design is highly scalable and has good performance in practice.

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Literature Survey

1.1 Grid Environments

Grid computing emerged as a framework for supporting complex operations over large datasets. Generally, grids enable the efficient sharing and management of computing resources for the purpose of performing large complex tasks. Grids have been defined as anything from batch schedulers to peer-to-peer (P2P) platforms.

Grid computing is an overloaded term. Depending on whom you talk to, it takes on different meanings. For the purpose of this discussion, we shall use the grid computing definition proposed by [129]: *Grid computing is any distributed cluster of computer resources that provide an environment for the sharing and managing of the resources for the distribution of tasks based on configurable service-level policies.* In the rest of this thesis we'll use the term cloud and grid computing, interchangeably.

Grid computing enables the harnessing of large numbers of processors working in parallel to solve computing problems that typically spread across various domains [115, 141]. Cloud data management [9, 10, 138], storage [17, 13] and security [147, 142] are some of the challenges recently receiving an increased interest. There also exist an increasing

number of large companies that are offering cloud computing infrastructure products and services [37, 12].

A grid fundamentally consists of two distinct parts, *compute* and *data*:

- **Compute grid** Provides the core resource and task management services for grid computing: sharing, management, and distribution of tasks based on configurable service-level polices.
- **Data grid** Provides the data management features to enable data integration, access, synchronization, and distribution.

Our focus in the following discussion is on data management on grids. Nowadays many data grid applications need to manage and process a huge amount of data distributed across multiple and heterogeneous grid nodes. Grids encourage the publication of data in a more open manner than is currently the case, and many e-Science projects have an urgent need to interconnect independently operated databases through a set of data access and integration services [11].

In the data grid area a set of services could address specific issues related to automatic data management, processing and integration aiming at both providing high performance and fully exploiting the grid infrastructure.

1.2 Data Federation Systems

The problem of combining heterogeneous data sources under a unified single query interface is an old one. The rapid use of databases after the 1960s led to the need to combine or merge existing repositories. This merging can be done at several levels in the database architecture. One popular approach is Data Warehousing, where data from several source repositories are extracted, transformed and loaded to a target repository, and then can be queried using a unified schema (i.e. of the target repository). This process is abbreviated

ETL and can be architecturally viewed as a tightly coupled approach because all source data reside in a single target repository at query time.

This tightly coupled approach suffers from a serious drawback related to the data synchronization between target and source repositories; every time an update is made to a source, the update should be propagated to the target. Another difficulty with this approach emerges when only a query interface is available at the source with no access to the whole actual data, this problem lately received more focus with the emerge of mashups and web services integration applications.

The recent trend in data integration has been to loosen the coupling between the data. The idea is to provide a uniform query interface over a target schema. The query against the target schema is then transformed into specialized queries over the source schemas.

A number of data integration systems have been proposed to address the problem of large-scale data sharing (e.g. [58], [49], [91], see also the survey by Halevy [65]). These systems support rich queries over large numbers of autonomous and heterogeneous data sources by making use of semantic relationships between the different source schemas and a *mediated schema*, that is designed globally. However, the mediated schema becomes a problem itself. First of all, it may be hard to come up with a single mediated schema that everyone agrees on. Moreover, all the access (querying) is done via a single point (the mediated schema). Furthermore, this architecture is not robust with respect to the changes in the source schemas. As a result, data integration systems based on mediated schemas are limited in supporting large-scale distributed and autonomous data sharing.

Peer Data Management Systems (PDMS), e.g., Piazza in [66], have been proposed to address the aforementioned problems and to offer an extensible and decentralized data sharing system. The requirements of the proposed integration architecture are, in principle, no different from these peer data management systems. The proposed data federation architecture emphasizes the use of tools and services that facilitate mappings among schemas

and generate the queries that are needed to access and integrate the data. In many respects our vision is similar in spirit to that of the SHARQ project [24] and the study in [137].

Some of the current research in data integration concerns the semantic integration problem. This problem is not about how to structure the architecture of the integration, but how to resolve semantic conflicts between heterogeneous data sources.

1.3 Data Federation in the Bioinformatics Domain

Biological data sources accessible through the internet are proliferating in an unprecedent manner and there is an increased need for architectures and tools that are capable of integrating information available from a variety of sources.

As of September 2006, the Gene Expression Omnibus (GEO) repository at the National Center for Biotechnology Information (NCBI) holds over 3.2 billion measurements deposited by more than 2000 laboratories from around the world [21]. Public centralized repositories are the state-of-the-art approach for scientists to collaborate and share their data.

Centralized repositories, like GEO or UCSC Genome Browser[79], are only partially serving such collaboration needs. Bioinformatics is a multidisciplinary field. As such, scientists from different domains may require a customized view or organization of data. For example, a specific schema definition may be suitable for a computer scientist. However, a biologist may be more comfortable with a different organization of the data. Moreover, the capabilities of relating such views, dynamic sharing, and evolution of data are challenges calling for novel web architectures. The proposed architecture is an attempt to address such challenges. It aims to transform public centralized web repositories into a decentralized one that could dynamically evolve, while providing more services. Groups of scientists can dynamically define new schemas, populate such schemas with data, update existing schemas/data, and define relations between existing schemas/data that others can use.

1.3.1 Challenges

There are numerous challenges that must be overcome when federating heterogeneous bioinformatics data sources. In the following, we start by discussing the major characteristics of those data sources and hence, highlight challenges that need to be resolved when trying to federate them.

- **Heterogeneous representation:** Similar data can be contained in several data sources but represented in a variety of different ways depending on the source. This heterogeneity include semantic, structural, naming, and content differences [134]. This highlights the challenge of dealing with schema complexity and structural differences among different sources. Moreover, each source may refer to the same semantic concept or field with its own identifier or term, which can lead to a semantic discrepancy between the many sources. The opposite may also happen, as some data sources may use the same term to refer to different semantic objects. Finally, the content differences involve sources that contain different data for the same semantic object, or that have some missing data, thus creating some possible inconsistencies between sources. This heterogeneity in representation leads to issues such as entity identification across sources and data inconsistency, redundancy and quality issues.

- **Data variety:** The data maintained by the current data sources cover several bioinformatics research fields. For example, typical data include gene sequences and expressions, disease information, molecular structures, micro-array data, protein sequences, structures and interactions, etc. Depending on how large or domain-specific the data sources are, they can store different types of data. Moreover, bioinformatics data can be characterized by many relationships between concepts and entities, which are sometimes difficult to identify formally since some of these concepts are abstract or in other cases they span several research fields. Also, the challenge is not because the quantity of data available in a data source is huge, but also the size of each data

item or record may itself be extremely large (e.g DNA sequences or 3-D protein structures). This differs, for example, from business data integration scenarios where there is usually no real need to handle the issue of very large data units.

- **Querying capabilities:** Individual sources provide their own user interface, which need to be learned in order to retrieve information. Additionally the sources often allow for only certain types of queries to be asked and hence protecting and preventing direct access to their data. These access restrictions force users and external systems to adapt and limit their queries to a certain form. Authors in [134] note that bioinformatics sources require biologists to master many different interfaces, moreover, some useful information cannot be retrieved because of query restrictions even though the data necessary to answer them is available in the data sources.

- **Autonomy of data sources:** Most of data sources operate autonomously, which means that they are free to modify their design and/or schema, remove some data without any prior "public" announcement, or occasionally block access to the source for maintenance or other purposes. Furthermore, they may not always be concerned by other sources referencing them or integration systems accessing them. Moreover, all sources are web-based and are therefore dependent on network traffic and availability. In addition to the sources being autonomous, the data is also dynamic and new discoveries will continuously modify the source content to reflect the new findings. So, the only way for an integration system to be certain about returning the latest information is to actually access the data sources at query time.

1.3.2 Variability among Federation Approaches

The existing systems for federating bioinformatics sources vary along several aspects, and we will discuss these aspects in the following:

- **Integration Objective:** This is concerned by the overall goal of the integration system. Some systems are "portal" based in that they aim to support an integrated browsing experience for the user (e.g. SRS [88], BioNavigator [127]). Others are more ambitious in that they take user queries and return results of running those queries on the appropriate sources (e.g. DiscoveryLink [62], TAMBIS [20]).

- **Inter-relations among data sources:** This is concerned about the assumptions made on the inter-relations between sources. Most systems assume that sources they are integrating are "complementary" in that they export different parts of the schema. Others also consider the possibility that sources may be overlapping (c.f. [6]) in which case aggregation and combination of information is required. Integrating complementary sources is sometimes called "horizontal integration" [134] while integrating the overlapping sources is called "vertical integration".

- **Data model:** This refers to the assumptions made by the integration system about the nature of the data being imported from the sources. Some systems have text models of data (e.g. SRS [88], BioNavigator [127]), while others have structured data models. In case of structured models, systems differ in terms of the specific model assumed including relational and object-relational data models (c.f. DiscoveryLink [62]) and nested semi-structured data models (e.g. XML and CPL [42]).

- **Usage types:** This is concerned about the type of usage scenarios that the system is designed to support. The systems that primarily support browsing need to assume little experience on the part of users. In contrast, systems that support user-queries need to assume some level of experience on the users part and in formulating queries. Some of these systems (e.g. [42]) assume queries to be formulated in specific languages, while others (c.f. [20]) provide interactive support for users in formulating queries.

- **User control level:** This addresses the extent to which the user has control over and is able to specify the particular sources that are needed to be used in answering queries. Some systems (e.g. [42]) require the user to select the appropriate sources to be used. Other sources (e.g. [20]) hard code specific parts of the integrated schema to specific sources.

The integration approaches used in the current systems can be classified first in terms of the data model they use text, structured data or linked records.

For systems that view sources as exporting mainly text, integration involves supporting keyword/text search across the sources. When the sources are viewed as exporting more structured data, there are two broad types of integration approaches, based on whether the data from the sources is warehoused or accessed on demand from the sources.

Finally, for systems that view sources as exporting linked sets of browsable content, integration involves supporting effective navigation across sources. Since the majority of systems use the (semi-)structured or linked record models, in the following, these approaches are discussed in more detail.

1.3.3 Warehouse-based Integration

The warehouse integration approach works by materializing the data from multiple data sources into a local warehouse and executing all queries on the data contained in the warehouse instead of the actual source data. Warehousing emphasizes data translation, as opposed to query translation in mediator-based integration [134] (discussed later).

In fact, warehousing requires that all the data imported from the sources be translated through data mapping to a standard representation before it is physically stored locally. Since warehousing relies less on the network to access the data, it is obvious that it helps eliminating various problems (e.g. network bottlenecks, low response times, and unavailability of sources).

Moreover, using materialized warehouses allows for the application of efficient query optimization strategies, since it can be performed locally [52, 41]. Another benefit in the warehouse integration approach is that it allows the system or the user to filter, modify, validate, and annotate the data retrieved from the sources [40, 68].

This approach however has some important drawbacks related to the data synchronization between the warehouse and source repositories; every time an update is made to a source, the update should be propagated to the warehouse. Warehouse integration must indeed regularly check throughout the underlying sources for new or updated data and then reflect those modifications on the local copy of the data [41]. Another difficulty with this approach emerges when only a query interface is available at the source with no access to the whole actual data, this problem lately received more focus with the emerge of mashups and web services integration applications.

1.3.4 Mediator-based Integration

Mediator-based integration concentrates on query translation. A mediator is a system that is responsible for reformulating (at runtime) a query given by a user on a single mediated schema into a query on the local schema of the underlying data sources. Unlike in the warehouse approach, none of the data in a mediator-based integration system is converted to a unique format according to a data translation mapping. Instead a different mapping is required to capture the relationship between the source descriptions and the mediator and thus allow queries on the mediator to be translated to queries on the data sources. Specifying this correspondence is a main step in creating a mediator, as it will affect both how difficult the query reformulation is and how easily new sources can be added to or removed from the integration system. The two main approaches for establishing the mapping between each source schema and the global schema are global-as-view (GAV) and local-as-view (LAV) [52, 86]. In the GAV approach, the mediator relations are directly written in terms of the source relations. In other words, each mediator relation is nothing

but a query over the data sources. The GAV approach facilitates query reformulation as it simply becomes a view unfolding process; however handling additions or removals of sources is much more difficult as it requires a modification of the mediator schema to take into account the changes.

In the LAV approach, every source relation is defined over the schema and relations of the mediator. Therefore, It is up to the individual sources to provide a description of their schema in terms of the global schema, making it very simple to add or remove sources but also complicating the query reformulation and processing role of the mediator.

Clearly both of these approaches have some cons and pros, but LAV is considered to be much more appropriate for large scale ad-hoc integration because of the low impact changes to the information sources have on the system maintenance, while GAV is preferred when the set of sources being integrated is stable and known in advance.

Several of the bioinformatics integration systems were developed before the advent of the mediated systems, and instead follow the federated database model. A federated database integration system consists of underlying sources which are autonomous components but which also cooperate to allow controlled access to their data. The study in [124] explains that federated integration can be seen as a middle-ground between no integration (where a user must query each source individually) and total integration (where a user can only query the sources through the integration system). In federated integration the schemas of the component sources are put together to form an integrated schema on which queries will be asked. Seen from this point, mediated systems could be seen as a loosely coupled versions of federated systems.

1.3.5 Peer Data Management System

Peer Data Management System ”PDMS” is a new class of data sharing tools that preserves semantics and rich query languages, but which facilitates ad hoc, decentralized sharing, and administration of data and defining of semantic relationships [66].

PDMS addresses two main problems in previous approaches: they typically require a comprehensive schema design before they can be used to store or share information and they are difficult to extend because schema evolution is heavyweight and may break backward compatibility. As a result, many small-scale data sharing tasks are more easily facilitated by non-database-oriented tools that have little support for semantics. The goal of the peer data management system (PDMS) is to address this need by proposing the use of a decentralized, easily extensible data management architecture in which any user can contribute new data, schema information, or even mappings between other peers' schemas. PDMSs represent a natural step beyond data integration systems, replacing their single logical schema with an interlinked collection of semantic mappings between peers' individual schemas.

1.3.6 Navigation-based Integration

The idea of link-based or navigation integration emerged from the fact that an increasing number of sources on the web require users to manually browse through several web pages and data sources in order to obtain the desired information [41]. In fact the main motive justifying this type of integration is that some sources provide the users with pages that would not or difficult to be accessible without this navigational approach. The specific paths constitute workflows in which the output of a source or tool is directed to the input of the next source until the requested information is reached [27]. Queries can be transformed into several path expressions that could each answer the query in a different way [105].

Navigational integration eliminates relational modeling of the data and instead applies a model where sources are defined as sets of pages with their interconnections, as well as additional information such as content, path constraints, and optional or mandatory input parameters [28, 86]. The study in [53] claims that this model effectively allows the representation of cases where the page containing the desired information is only reachable through a particular navigation path. The study in [83] explores the path-based approach

by analyzing paths between biological sources. The observation was that multiple physical paths can link two sources. Therefore, the goal is to determine properties of links between sources and use these properties to identify the best of several potential execution paths that can answer a given query.

1.3.7 Discussion on Integration Approaches

As previously discussed, the warehouse approach can provide two clear advantages. First, it simplifies query optimization and processing by storing the data locally according to a single global schema. Second, it enables users to add their own annotations to some stored data and specify some filtering conditions to clean the data as it is stored locally. Although this would indeed be a definitive improvement for the user of the system, it is still unclear how this process could be achieved efficiently, and more specifically how the data could effectively be validated or modified without requiring costly and time consuming human intervention, as well as extensive domain expertise. The data retrieved and integrated in the warehouse will indeed eventually have to be converted into a warehouse-specific format. Furthermore, as mentioned earlier, data warehousing in general must still face the vast problem of handling updates in the data sources, which here would be an even greater challenge as the data contained in the warehouse may be modified and annotated, and therefore always different from the data in the underlying sources. It is however interesting to note that despite those negative aspects, authors in [68] have suggested that a large curated warehouse was the only effective approach, given the pervasive data inconsistency across sources.

The GAV and LAV approaches were discussed for mediator-based integration, it is interesting to note that they are not frequently implemented in the biological integration systems. A reason may simply be that biologists started working on biological databases and repositories long before web-based data integration became a research field in Computer Science [69]. Initially, most systems were either warehouses or federated databases, which

have survived and evolved since. Only relatively recently has the field lead to wrapper-oriented or navigational approaches. The concept of navigational integration has in fact not yet established itself as a true alternative to the other, more common integration approaches. The study in [41] even doubt that link-driven integration is pertinent to bioinformatics integration because it does not offer enough querying functionality and because it is not well adapted to the extent of the data available and to the continuously changing sources. However, path-based optimization of queries seems like a promising direction.

Most of the currently widely used integration systems (like TAMBIS and K2 [40]) only address the horizontal dimension of data integration. In integrating only sources that have complementary data, an integration system does not take into account the potential overlapping aspect of sources or the probable incompleteness of some sources. Restricting the integration process to simply combining data from sources that contain different types of information limits the capability of a system, especially in terms of reliability and completeness.

A purely horizontal integration system cannot address important issues related to effectiveness and efficiency. Aggregation of information and sources is also necessary. Considering the possibility of having several candidate sources for the same mediator relation would essentially allow a system to address these issues. DiscoveryLink makes an attempt to solve the problem of selecting between several potential sources by using the estimated query processing cost given by the individual wrappers, although the overlap and coverage point of view of optimization and source selection is not considered.

The concept of the peer data management system emphasizes not only an ad hoc, scalable, distributed peer-to-peer computing environment, which is compelling from a distributed systems perspective, but it provides an easily extensible, decentralized environment for sharing data with rich semantics. This is in contrast to data integration systems, which have a centralized mediated schema and administrator and which can impede small, point-to-point collaborations. It also complements the knowledge representation work of

the Semantic Web by providing a mechanism for translating between different ontologies data representations. Our proposed data federation architecture builds on the concepts of peer data management, and identifies the essential components for realizing such systems. The proposed architecture emphasizes the use of tools and services that facilitate mappings among schemas, combining schemas into unified integrated non-redundant representation and generating the queries that are needed to access and integrate the data.

1.4 The Schema Matching and Integration Problems

Some of the current research in data integration is concerned with the problem of semantic integration. This problem is not mainly about the architectural structure of the integration process, on the other hand, it is specifically concerned with the problem of resolving semantic conflicts between heterogeneous data sources.

The objective of schema integration is to find a unified and nonredundant representation of the data, used to simplify the access to heterogeneous data sources. In artificial intelligence, this can be regarded as the problem of integrating independently developed ontologies into a single ontology [118].

Since the schemas are independently developed, they often have different terminology and structure. This can be easily attributed when the schemas are from different domains. But, it can also occur even if they model the same real world domain, merely because they were developed by different people in different contexts.

A first step in schema integration is to identify the inter-schema relationships. This is a process of *schema matching*. Once these matching elements are identified, they can used to construct a unified integrated schema. During the integration process or sometimes as a separate step, queries can be created that permit translation of data from the original source schemas into the target integrated schema. A variation of the schema integration problem is to integrate an independently developed schema with a given conceptual schema, which re-

quires reconciling the terminology and structure of the two schemas, which again involves schema matching. In Section 1.4.1 we'll review the schema matching problem, then we'll build on this discussion to formulate our problem and introduce our approach for schema integration in Section 1.4.2.

1.4.1 Schema Matching

One of the basic operations in data integration is the process of matching concepts describing the meaning of data in heterogeneous distributed data sources. Due to the cognitive complexity of this matching process, it has traditionally been performed by human experts (e.g., database analysts).

Schema matching is a fundamental operation in the manipulation of schema information, which takes two schemas as input and produces a mapping/correspondances between elements of the two schemas that correspond semantically to each other [118]. Schema matching plays a central role in various applications, such as web services data integration, schema integration, data warehousing and database design.

Traditionally, schema matching was typically performed manually, in some cases supported by a graphical user interface. It is clear that manually specifying schema matching is a tiring, time-consuming, error-prone, and therefore expensive process. This is a growing problem given the rapidly increasing number and complexity of data sources. Moreover, the level of effort is at least linear in the number of matches to be performed, even worse than linear if we need to evaluate each match in the context of other possible matches of the same elements. Moreover, introduction of the Semantic Web vision and the shift towards machine-understandable web resources have manifested the importance of automatic matching between sets of elements. So, a faster and less labor-intensive matching approach is needed and this requires automated support for schema matching. Generally, it is not possible to fully determine automatically all matches between two schemas, because most schemas have some semantics that affects the matching process but is not formally

or explicitly expressed. The schema matching should therefore determine match candidates, which the user can accept, reject or change. Moreover, sometimes the user could be involved in specifying matches for elements for which the system was unable to find satisfactory match candidates.

Schema matching methods can be classified based on various perspectives as described in the following [118]:

- **Instance vs schema:** matching approaches can consider instance data (i.e., data values) or only schema-level information (i.e., metadata).
- **Element vs structure matching:** match can be performed for individual schema elements, such as attributes, or for combinations of elements, such as complex schema structures.
- **Language vs constraint:** a matcher can use a linguistic-based approach (e.g., based on names and textual descriptions of schema elements) or a constraint-based approach (e.g., based on data types, keys and relationships).
- **Matching cardinality:** the overall match result may relate one or more elements of one schema to one or more elements of the other, giving four cases: 1:1, 1:n, n:1, n:m. In addition, each mapping element may interrelate one or more elements of the two schemas.
- **Auxiliary information:** most matchers rely not only on the input schemas but also on auxiliary information, such as ontologies, dictionaries, global schemas, previous matching decisions, and user input.

Note that the classification does not distinguish between different types of schemas (e.g., relational, XML, object-oriented, etc.) and their internal representation, because schema matching algorithms depend mostly on the kind of information they exploit, not on its representation.

The above classification is concerned with individual matching techniques. Sometimes it is useful to use multiple matchers, therefore, it is important to differentiate two sub-problems. First, there is the realization of individual matchers, each of which computes a matching based on a single criterion. Second, there is the combination of individual matchers, either by using multiple matching criteria within an integrated hybrid matcher or by combining multiple match results produced by different match algorithms within a composite matcher.

In the following, we discuss the main alternatives according to the above classification criteria. We discuss schema-based matching followed by a discussion of instance-based matching.

Schema-based Matching

Schema-based matchers only consider schema information, not instance data. The available information includes the usual properties of schema elements, such as name, description, data type, relationship types (part-of, is-a, etc.), constraints, and schema structure. In general, a matcher will find multiple match candidates. For each candidate, it is customary to estimate the degree of similarity by a normalized numeric value in the range 0...1, in order to identify the best match candidates (as in [107], [106], [23], [46] and [32]).

We first discuss the main alternatives for match granularity and match cardinality. Then we cover linguistic and constraint-based matchers.

- **Match granularity:** There are two main alternatives for the granularity of the matching, *element-level* and *structure-level* matching. For each element of the first schema, element-level matching determines the matching elements in the second input schema. In the simplest case, only elements at the finest level of granularity are considered, which we call the atomic level, such as attributes in an XML schema or columns in a relational schema. On the other hand, structure-level matching refers to matching

combinations of elements that appear together in a structure. A range of cases is possible, depending on how complete and accurate a match of the structure is required.

- **Match cardinality:** An element in the schema can participate in zero or more mapping elements of the match result between the two input schemas. Moreover, within an individual mapping element, one or more elements in one schema can match one or more elements in the other schema. Thus, we have the usual relationship cardinalities, namely 1:1 and the set-oriented cases 1:n, n:1, and n:m, between matching elements. When matching multiple elements at a time, expressions are used to specify how these elements are related. Previous work has mostly concentrated on such 1:1 matches because of the difficulty of automatically determining the mapping expressions in the other cases. Most existing approaches map each element of one schema to the element of the other schema with highest similarity. More work is needed to explore more sophisticated criteria for generating local and global n:1 and n:m mappings, which are currently hardly treated.

- **Linguistic approaches:** linguistic or language-based matchers use names and text to find semantically similar schema elements. We shall discuss two schema-based approaches, *name matching* and *description matching*. **Name matching** Name matching matches schema elements with equal or similar names. Similarity of names can be defined and measured in several ways, including: 1) Equality of names (an important variation is the equality of names from the same XML namespace), 2) equality of canonical name representations after preprocessing (e.g. stemming), 3) equality of synonyms, 4) equality of hypernyms, 5) similarity of names based on common substrings, edit distance, pronunciation, etc., and 6) user-provided name matches. **Description matching** Schemas often contain comments in natural language to express the intended semantics of schema elements. These comments can also be linguistically evaluated to determine the similarity between schema elements

- **Constraint-based approaches:** Schemas often contain constraints to define data types and value ranges, uniqueness, options, relationship types and cardinalities, etc. If both input schemas contain such information, it can be used by the matching module to determine the similarity of schema elements [84]. For example, similarity can be based on the equivalence of data types and domains, of key characteristics (e.g., unique, primary, foreign), of relationship cardinality (e.g., 1:1 relationships).

Instance-based Matching

Instance-based data can give important insight into the contents and meaning of schema elements. This is extremely useful when schema information is limited, as is often the case for semi-structured data, or in the extreme case, when no schema is given. Even when substantial schema information is available, the use of instance-based matching can be valuable to uncover incorrect interpretations of schema information. For example, it can help disambiguate between equally evaluated schema-based matches by choosing to match the elements whose instances are more similar. Most of the approaches discussed previously for schema-based matching can be applied to instance-based matching.

Several tools for automated schema matching, such as GLUE [47] and OntoBuilder [57], have been developed in recent years. Given two data schemata (e.g., two sets of attributes), these tools output a single mapping from elements of one schema to elements of the other. The outputted mapping is considered to be the best of all possible mappings between these schemata.

Although these tools comprise a significant step towards fulfilling the vision of automated schema matching, it has become obvious that the user must accept a degree of imperfection in this process [56]. A main reason for this is the enormous ambiguity and heterogeneity of data description concepts: It is unrealistic to expect a single mapping engine to identify the correct mapping for any possible concept in a set. Another reason is

that “the syntactic representation of schemas and data do not completely convey the semantics of different databases” [95]; i.e., the description of a concept in a schema can be semantically misleading. Therefore, managing uncertainty in schema matching has been recognized as the next issue on the research agenda in the context of data integration [90].

The study in [55] offers an uncertainty management tool, extending the practice in schema matching by using top-K schema mappings rather than a single best mapping. This is a natural extension of existing methods (which can be considered to fall into the top-1 category), taking into account the uncertainty described above.

1.4.2 Schema Integration

The input to integration is a set of source schemas that relate to each other through correspondences or constraints. The output is a consolidated target schema that constitutes a nonredundant unified representation of all the data.

Schema integration has been an active research field for a long period of time and continues to be a challenge in practice [22, 26, 96, 128, 111, 34, 112]. This problem lies at the core of many metadata applications, such as view integration, mediated schema creation, and ontology merging. The spectrum of applications extends to web-service integration, mashups and distributed web architectures [115].

Although today the process of integrating schemas is partially automated, it is still labor-intensive. In order to reduce the amount of manual intervention that is required from users, we need to modify or avoid parts of the integration process that unnecessarily increase the load on users. Let us follow the steps that generally need to take place while combining two input schemas.

First, the input schemas are run through one or more schema matching algorithms that return *correspondences* between the elements of the schemas. Such correspondences typically have weights reflecting the confidence of the matchers that the two elements are similar or have overlapping semantics. Moreover, the weights in each direction can be

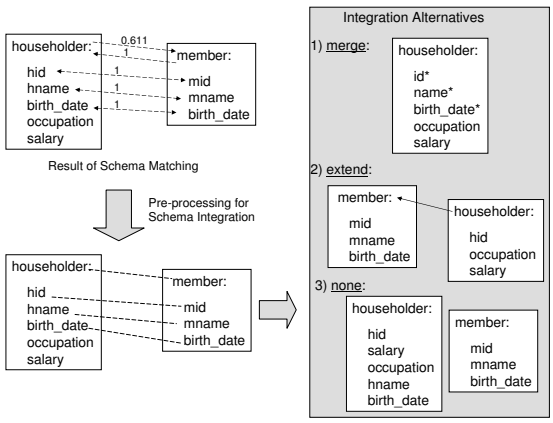

Figure 1.1: State of the art.

different; thus, an element A can be similar to (or covered by) an element B with weight w, while the element B is similar to (or covered by) element A with weight w', where w and w' are not necessarily equal. We say in such situation that there are two *directed and weighted correspondences* between elements A and B. In a second step, all the directed correspondences between two elements[1] are merged (by some aggregation process) into one undirected correspondence with one aggregated weight. Correspondences for which the aggregated weight is above a threshold are kept, while the rest are discarded. Moreover, after the above pruning step, the weights of the remaining set of correspondences are typically themselves discarded. In the third step, several alternatives for combining the input schemas are available, based on the surviving correspondences, and schema integration tools provide interactive means for the users to select a desired integrated schema.

Consider the naive example in Figure 1.1. The two simple input schemas describe the structure of two elements: *householder* and *member*. These schemas are run through one or more matching algorithms. For simplicity, assume in this example that atomic elements that match are assigned correspondences weighted with a similarity of 1, in both directions. Correspondences that have weight 0 are not shown. The schema matching algorithm then calculates the overall similarities between the non-atomic elements, by aggregating

[1]There could be more than two if there are multiple schema matchers.

in some way the similarities of the sub-elements. In this example, all the sub-elements of *member* match the sub-elements of *householder*, and therefore *member* is covered entirely by *householder*. Thus, we can conclude that there is a directed correspondence *member* $\rightarrow$ *householder* with the similarity/weight of 1. On the other hand, the element *householder* contains some sub-elements that are unique, and as a result the similarity of the directed correspondence *householder* $\rightarrow$ *member* is less than 1.[2] One can see that the weights of the derived correspondences give valuable hints about the coverage of one element by another. Such information is more refined than just saying whether two elements match or not. In particular, it can suggest that one element is likely to be an extension or represents a sub-concept of the other element.

In the second step, the results of the matching algorithm are transformed into the undirected correspondences used for the generation of the integrated schema. There are several ways to combine the elements of the two schemas, but in this simplified example there are three natural choices: (1) merge the two root elements *householder* and *member* into one, (2) introduce an extension relationship (or reference) that will say that *householder* is an extension of *member*, or (3) do not combine the two elements at all and just take the union of them. This is a simple example; in reality schemas are larger and there are more complex relationships between elements. Consequently, the space of *possible* candidate schemas that can result from combining the input schemas can be quite large.

The third step in the schema integration process is then concerned with identifying the "best" integrated schema among these alternatives. In most methods, the alternatives are not created explicitly in the system, and the user must "drive" the generation of *one* integrated structure. A good example of such system is described in [111]; in their method, the expert provides a "template" of the integrated schema (also called a mapping model) that effectively specifies the structure of the merged schema and provides the basis for accumulating all the attributes and the relationships from the input schemas. A different kind

[2] Section 3.3 will show one method for calculating the similarity measure for complex elements

of method, that is based on the explicit identification of the alternative schema structures, is described in [34]. In their system, a user can systematically explore the alternatives and narrow down, in an interactive way, the desired integrated schema. The advantage of such method is that it is based on a systematic enumeration of the design choices and provides more information to a user. The disadvantage is that it still relies heavily on the user to explore the available choices and decide which ones are better.

Overview of our approach In this thesis we address the above shortcomings in the schema integration process as follows. (See also Figure 1.2.) First, we keep all the information generated by the matching algorithms. In particular, we make use of both the direction and the weights associated with the correspondences between elements. This information enables us to define more refined relationships on the integrated schema. More importantly, this information enables us to *rank* the integrated schemas, by giving higher priority to schemas that combine the concepts with higher similarity or coverage. Based on this ranking, we devise a polynomial time, efficient algorithm for the generation of the top-k integrated schemas. As a consequence, the resulting system avoids the generation of the unlikely schemas and therefore minimizes the user effort. In the extreme, the process can be entirely automated by generating the single schema that is the best according with our ranking.

We also show experimentally that the algorithm performs well in practice, on real schemas, and moreover it generates the "expected" schemas.

Our method uses an extension on the framework of [33, 34] for schema enumeration, where we replace the user-driven exploration of the space of schemas with top-k generation. As in [34], we use graphs of *concepts* with has relationships to represent, at a higher-level of abstraction, XML and relational schemas. A concept of a schema is a relation name with an associated set of attributes and, intuitively, represents one category of data (an entity type) that can exist according to that schema (e.g., an "employee", a "department", an "address", etc.). Concepts in a schema may have references to other concepts in the

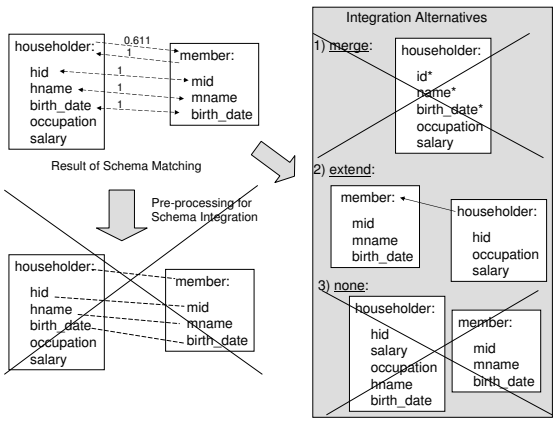

Figure 1.2: Our solution: keeps more information, outputs the more likely schemas.

schema and these references are captured by has edges (e.g., "employee" contains a has edge to "department", etc.).

We note that schema matching techniques have been extensively studied [118, 93, 25]. Our method is complementary to schema matching, since it uses the outcome of schema matching. Furthermore, the emphasis here is on using directed similarities rather than the more common undirected similarities.

1.5 Bioinformatics Applications

A problem facing many bioinformatics researchers today is the aggregation and analysis of vast amounts of data produced by large scale projects from various laboratories around the world. Depositing such data into centralized web-based repositories (e.g. NCBI, UCSC Genome Browser) is the common approach. However, the distributed nature of the data, its growth rate, and increased collaborative needs represent real challenges calling for novel decentralized web architectures.

We study the requirements for realizing a web services-based data federation architecture for bioinformatics applications. Based on collaborations with bioinformatics researchers, several domain specific data federation challenges and needs are identified. We

address such challenges and provide an architecture that incorporates a series of utility services. These address issues like automatic workflow composition, domain semantics, and the distributed nature of the data. It also incorporates a series of data-oriented services that facilitate the actual integration of data. The proposed design, services, and usage scenarios are studied and discussed. We demonstrate how our architecture can be leveraged for real-world bioinformatics problems; specifically we address the analysis of nucleosome exclusion regions across the human genome to study its role and importance as a gene regulation mechanism and how it is correlated to tissue specificity, gene density and other features.

Nucleosomes are the basic structural units of eukaryotic chromatin, and they play a significant role in regulating gene expression. Specific DNA sequence patterns are known, from empirical and theoretical studies, to influence DNA bending and flexibility, and have been shown to exclude nucleosomes. A whole genome localization of these patterns, and their analysis, can add important insights on the gene regulation mechanisms that depend upon the structure of chromatin in and around a gene.

A whole genome annotation for nucleosome exclusion regions (NXRegions) was carried out on the human genome. Nucleosome exclusion scores (NXScores) were calculated individually for each nucleotide, giving a measure of how likely a specific nucleotide and its immediate neighborhood would impair DNA bending and, consequently, exclude nucleosomes. The resulting annotations were correlated with 19055 gene expression profiles. We developed a new method based on Grubbs' outliers test for ranking genes based on their tissue specificity, and correlated this ranking with NXScores. The results show a strong correlation between tissue specificity of a gene and the propensity of its promoter to exclude nucleosomes (the promoter region was taken as -1500 to $+500$ bp from the RefSeq-annotated transcription start site). In addition, NXScores correlated well with gene density, gene expression levels, and DNaseI hypersensitive sites. Nucleosome exclusion patterns are correlated with various factors that regulate gene expression, which empha-

sizes the need to include chromatin structural parameters in experimental analysis of gene expression.

1.6 A Cloud Data Processing Abstraction Layer

Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction [35]. Cloud computing enables the harnessing of large numbers of processors working in parallel to solve computing problems that typically spread across various domains [115, 45, 141]. Cloud data management [9, 10, 138], storage [17, 13] and security [147, 142] are some of the challenges recently receiving an increased interest. There also exist an increasing number of large companies that are offering cloud computing infrastructure products and services [37, 12].

Cloud computing has lately received wide interest, and along with this interest has come several advancements in the tools for programming them. MapReduce (MR) is one such tool, an attractive option to many programmers because it provides a simple model through which users are able to express relatively sophisticated distributed programs [130].

MapReduce is a programming paradigm to perform parallel computations over distributed and very large data sets. There are numerous open source and commercial available implementations. The most popular MR system is Hadoop, an open-source project under development by Yahoo! and the Apache Software Foundation [18]. The details of MapReduce are described in [43, 44]. In the rest of this study, the focus will be on Hadoop implementation of MR. In MR basically the computation takes a set of input key/value pairs, and produces a set of output key/value pairs. The user of the MapReduce library expresses the computation as two functions: Map and Reduce. Map, written by the user, takes an input pair and produces a set of intermediate key/value pairs. The MapReduce

library groups together all intermediate values associated with the same intermediate key and passes them to the Reduce function. The Reduce function, also written by the user, accepts an intermediate key and a set of values for that key. It merges together these values to form a possibly smaller set of values. Accordingly, the Map and Reduce functions supplied by the user have the following associated types:

$$map(k1, v1) \rightarrow list(k2, v2) \tag{1.1}$$

$$reduce(k2, list(v2)) \rightarrow list(k3, v3) \tag{1.2}$$

Figure 1.3 illustrates the basic dataflow in a MapReduce job. The inputs are divided into several input splits, an "InputFormat" typically decides on the number of splits and how the splitting is performed. For every input split, a Mapper (i.e., Map task) is assigned, where a reader is responsible for reading the split and generating the stream of input key/value pairs to be fed to the Mapper. The Map function is called once for every input pair and can generate zero or more intermediate key/value pairs. Every intermediate pair is forwarded to a partitioner; all Map tasks share the same partitioning function. The partitioner decides on what Reducer (i.e., Reduce task) this pair should be forwarded to. The Map task maintains a buffer for every configured Reducer. The intermediate pairs are saved into this buffer, and when the buffer is filled-up, its contents are flushed into files on disk. In a following shuffling phase, the MapReduce framework transfers these intermediate files to their corresponding Reducers. The files for a Reduce task are merged and then sorted using the intermediate key, the main function of the sorting is to group the key/value pairs for every distinct key into a continuous sequence on disk, as a result, each Reduce function call can be fed an intermediate distinct key and an iterator over the corresponding values on disk. Iterating over the values on disk eliminates any memory constraints on the Reduce-side. The Reduce function can generate zero or more output key/value pairs, these pairs are fed to a writer as part of the "OutputFormat" to write these pairs in the correct format to the

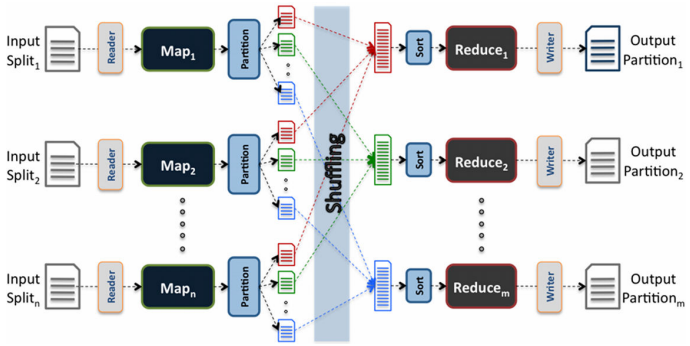

Figure 1.3: Basic dataflow within a single MapRecduce job.

corresponding output part. The above brief discussion focused on the basic MapReduce components, the interested reader can refer to [144] for more details.

1.6.1 Motivation

MapReduce offers a highly distributed programming paradigm that is efficient and scalable. However, using the low-level MapReduce for general data processing tasks poses the problem of developing, maintaining and reusing custom low-level user code. In MapReduce, the job is composed generally by specifying two functions/classes; a Mapper and a Reducer. For nontrivial applications, the code for this Mapper and Reducer can get really large and complex, and users face the problem of maintaining and reusing this code. Another factor that contributes to the increased complexity of Mappers and Reducers is the fact that each job is required to read its inputs from disk and write the results back to disk. The necessity to avoid frequent disk access promotes a tendency from MapReduce programmers to write Mappers and Reducers with aggregated and complex semantics. This coarse granularity of the definitions for the Mappers and Reducers forfeits the appealing simplicity proposed by the MapReduce model.

Several high-level frameworks have emerged to address this problem; Pig [103], Hive [138], Cascading [31] and Jaql [74] are examples for such frameworks. Hive provides a declarative query language based on SQL. Using Hive, users can express their problem

semantics in an SQL-like language and Hive takes care of translating these semantics into MapReduce constructs. Hive provides a convenient solution for users familiar with SQL.

On the other hand, Pig provides a procedural workflow language, namely, Pig Latin. The user uses Pig Latin constructs to formulate the problem, and the Pig system translates these semantics to MapReduce. Pig is suitable for users who are more comfortable in expressing their data processing problems in a procedural fashion (as opposed to the declarative way provided by Hive SQL).

Cascading provides an API for constructing data processing workflows. The objective is to let the developer quickly assemble complex distributed processes without thinking in MapReduce. Cascading introduces five core components: Tuple, Pipe, Tap, Flow, and Cascade. Users use these components to construct the workflow and Cascading translates them to MapReduce. Cascading also provides a scripting interface to perform these operations.

Jaql is a query language designed for Javascript Object Notation (JSON), its core features include semi-structured data modeling, user extensibility and parallelism. Jaql is a functional query language that provides users with a simple, declarative syntax to do things like filter, join, and group JSON data. MapReduce is used to achieve the required parallelism. The user formulates the query using Jaql constructs and the query is then translated to MapReduce for further execution.

1.6.2 A common feature of existing solutions

The aforementioned systems share a top-down approach, where a high-level language (or additionally an API in case of Cascading) is used to describe the problem semantics and the system takes care of "translating" this problem definition into MapReduce. The design of the high-level language and the mapping of the language constructs to MapReduce are critical aspects differentiating such systems.

Each of these high-level systems is limited by the expressive power of its language and the mechanism that translates the language constructs to MapReduce. Additionally, the

translation layer usually becomes a source of ambiguity and imposes a control limitation on the user-side; the user doesn't have real low-level control over the details of these constructed MapReduce jobs. To exercise more control, users may decide to directly write their own Mappers and Reducers, and hence, may again face the maintenance and reusability problems mentioned earlier.

Additionally, the aforementioned systems, assume that the user does not want to think and formulate the problem using MapReduce constructs (i.e., Map and Reduce functions), and hence, they provide a high-level language/abstraction with different semantics and constructs. Although this is true in some cases, there are still considerable number of MapReduce users who prefer to think in MapReduce and write their own Mappers and Reducers.

Accordingly, there is a need for a refined MapReduce model, which gives the user the ability to write their own Mappers and Reducers, but, at the same time, solve code maintenance and reusability problems.

1.6.3 LEGOS Approach

Refined MapReduce Job definition: MR-LEGOS offers an explicit model for composing Mappers and Reducers from simpler components; "Maplets" and "Reducelets". Maplets and Reducelets are standard Mappers and Reducers. This composition of Maplets and Reducelets can be viewed as defining a micro-workflow inside the MapReduce job. As a result, complex job semantics can be pushed away from Mapper and Reducer code and be explicitly defined in the encompassing micro-workflow provided by MR-LEGOS.

Using MR-LEGOS, MapReduce programmers will be encouraged to write simple Maplets and Reducelets that can be easily maintained, reused, and shared. Significant part of the complexity of the job semantics can be defined using the micro-workflow as later demonstrated using a data processing example in Section 5.2. The refined model is only limited by the expressive power of the underlying MapReduce system, based on the fact that the user has full control over the MapReduce job definition. MR-LEGOS only offers a logical

organization of such definitions. A MR-LEGOS job, in its simplest form, can be reduced to the original MapReduce definition, if the user defines a MR-LEGOS job that is merely composed of a single Maplet and Reducelet.

Data Processing Maplets and Reducelets: Using the MR-LEGOS refined model, users can share common reusable sets of domain specific Maplets and Reducelets. For example, in this thesis we share the experience of building a data processing set of Maplets and Reducelets, which are mostly relational data processing operators, in addition to a growing set of Maplets and Reducelets that were found to be useful and common across several data processing problems. We also believe this experience could be generalized in different application domains.

Chapter 2

Bioinformatics Data Federation System Design (The BioFederator)

This proposed design presents a life sciences research architecture hosted by the IBM sponsored LA Grid project[3]. LA Grid, an international grid research community, involves collaborations between a number of IBM research centers and many universities from the USA, Latin America, and Spain. It aims to facilitate collaborative research and development amongst participating institutions.

The presented architecture "The BioFederator" utilizes existing technologies and research results, in an attempt to make progress toward the ever complicated data integration problem in bioinformatics. Specifically, we have componentized Clio [63], a Java application for schema mapping and data transformation, into several key data-oriented services (i.e. schema mapping, query generation, query rewriting, query execution, and XML transformation), each wrapped as a web service.

As a proof-of-concept approach, we utilized Taverna [72], a bioinformatics workflow tool, to chain together the web services above and solve concrete integration problems in the bioinformatics domain. Using Taverna, one can manually compose a workflow by chaining together a group of web services. However, automatic workflow composition together with capturing domain semantics are two indispensable challenges for a practical bioinformatics data federation system on the web.

[3]Latin American Grid (LAGrid) – http://latinamericangrid.org/index.php

To address such challenges we augmented the data-oriented services with a collection of utility services (i.e., coordinator, semantic catalog, repository, and synchronization). The objective is to help automate the process of workflow composition, capture the required domain semantics and address the distributed nature of the data. The BioFederator is an attempt to define the essential components for providing an automated, domain-specific, modular, and decentralized data federation system on the web. Nodes of the system can provide data and services (data-oriented or utility services) or a combination of both. Users are able to contribute new data, define new relationships among existing schemas and data sources, relate data to domain-specific concepts, and construct new schemas that others can use.

The main contributions of this system are as follows. First, we present a set of data-oriented web services that we believe are essential for realizing a high-level declarative data federation system on the web. The study details the design, interface, and interaction among those services. Second, we identify a set of essential components and utility services that address the distributed nature of the data, capture domain semantics, and facilitate automatic workflow composition. Finally, in collaboration with bioinformatics researchers within the LAGrid community, we demonstrate how our architecture can be leveraged on a real-world bioinformatics problems.

2.1 Background

As of September 2006, the Gene Expression Omnibus (GEO) repository at the National Center for Biotechnology Information (NCBI) holds over 3.2 billion measurements deposited by more than 2000 laboratories from around the world [21]. Public centralized repositories are the state-of-the-art approach for scientists to collaborate and share their data.

Centralized repositories, like GEO or UCSC Genome Browser[79], are only partially serving such collaboration needs. Bioinformatics is a multidisciplinary field. As such, sci-

entists from different domains may require a customized view or organization of data. For example, a specific schema definition may be suitable for a computer scientist. However, a biologist may be more comfortable with a different organization of the data. Moreover, the capabilities of relating such views, dynamic sharing, and evolution of data are challenges calling for novel web architectures. The BioFederator is an attempt to address such challenges. It aims to transform public centralized web repositories into a decentralized one that could dynamically evolve, while providing more services. Groups of scientists can dynamically define new schemas, populate such schemas with data, update existing schemas/data, and define relations between existing schemas/data that others can use.

The BioFederator team involves collaborators from biomedical and genetics domains. The objective of the collaboration is to better understand domain-specific problems and needs, and to ensure that realistic bioinformatics scenarios are addressed. A number of bioinformatics studies exhibit the need for integrating data from multiple sources. The study in [122] mentions some examples. Pharmacogenomics is another example for an emerging branch dealing with the influence of genetic variation on drug response in patients, promising the advent of "personalized medicine" in which drugs and drug combinations are optimized for each individual's unique genetic makeup. To make such "personalized" medical decisions, information from multiple heterogeneous data sources needs to be incorporated. Examples are OMIM [92] and dbSNP [126] from NCBI, TRANSFAC database from BioBase [78], and PharmGKB [82].

Figure 2.1 shows an example study that aims to understand the rates of genes expressions in different tissues and correlate these expression profiles with active transcription factors and their binding sites. The data required for this study is distributed among multiple heterogenous sources (e.g., UCSC Genome Browser, GNF SymAtlas [132] and TRANSFAC). The figure shows how Clio mapping technology can be used to provide a high-level definition for mappings between the source and target schemas. In particular, a graphical user interface allows entering correspondences that relate schema elements. The

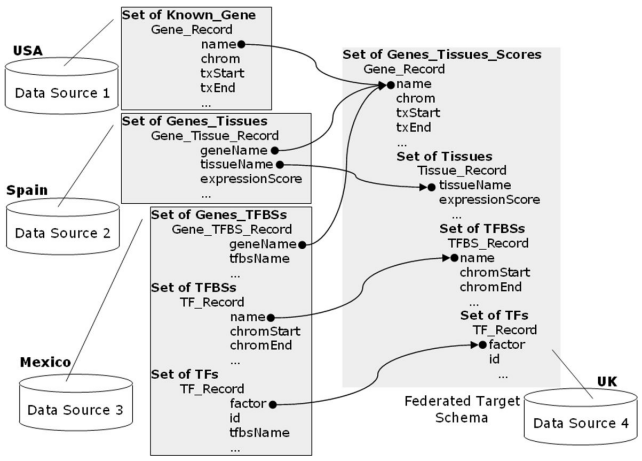

Figure 2.1: High-level mapping definition using Clio

schema mapping service takes the correspondences as input and computes a more precise mapping specification (based on the schema information, constraints, etc). This mapping specification can subsequently be used by other services. More details on the specific data services are provided in the *Architecture and Services* section, and this figure will be revisited in more detail in the *Usage Scenarios* section.

It should be noted that the BioFederator focuses on bioinformatics data integration applications. However, a typical bioinformatics research involves both computational and data driven aspects. While data driven processes may extract data from various sources, the computational processes would process the data through algorithms for pattern matching, sequence alignment, clustering etc. A computational application is discussed in Chapter 4 involving the study and analysis of nucleosome exclusion regions in the human genome.

2.2 Architecture and Services

The system is built from a collection of nodes. A node can be any device or computer having the required web services deployed. Nodes can be dynamically added or removed, and the system adapts to deliver quality of service requirements. Figure 2.2 depicts the architecture of typical BioFederator nodes. Each node contains six components/services, namely:

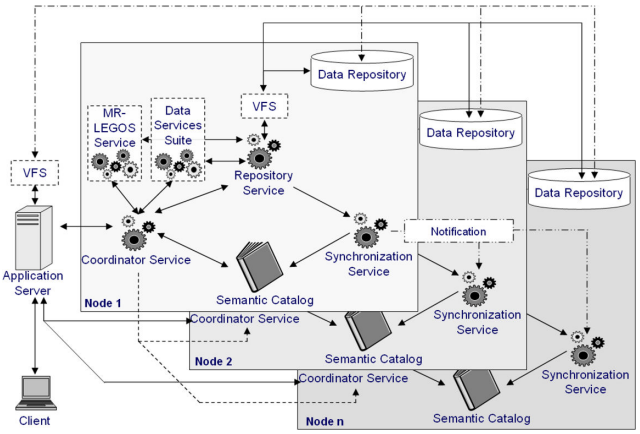

Figure 2.2: BioFederator nodes architecture

the *coordinator service*, the *semantic catalog*, the *repository service*, a local *data repository*, the *synchronization service*, and the *data services suite*. To support system portability, all components and services are built using Java and XML. The proposed system architecture supports distributed storage and manipulation of data. One or more application servers can connect users to the BioFederator. The application server can initiate multiple concurrent requests (by contacting multiple nodes). However, for every request, the application server designates a particular node as the "master" node. Any node is capable of performing the "master" role. The master can distribute the required operations among many other nodes and is also responsible for coordinating, collecting, and merging results from "slave" nodes.

Coordinator Service: This service is the front-end interaction point for a given BioFederator node. It maintains a list of all registered nodes in the whole system, in addition to locally and remotely deployed services. It can access the metadata maintained in the local semantic catalog (discussed later in detail). Coordinator services are key elements for automated workflow construction because they are responsible for decisions involving forwarding, splitting, or directly handling a received request.

Upon receiving a request, the coordinator needs to access the semantic catalog to retrieve information about data needed/involved in the request. Based on metadata and infor-

mation about registered nodes and deployed services, the coordinator can make a decision on how to handle the request (i.e., forward, split or directly handle).

The coordinator maintains a set of rules to help make such decisions, the efficiency and accuracy in defining such rules is crucial for correct and efficient system performance. The rules are a function of the quality of service, load balancing, and optimization requirements. While a system could function well using limited rules, its performance could be enhanced by adding and tuning rules. For example, a simple rule is to forward the request to the first node that has the required services deployed. However, a better rule is to incorporate the location of the data. From our experience, fine tuning rules may result in complex but efficient workflows of services. Policies like query distribution and data materialization are initiated by the coordinator service.

Semantic Catalog: This service provides a domain-specific organization of the data. Figure 2.3 depicts the structure of an illustrative subset of the semantic catalog. The BioFederator does not require a fixed structure for the semantic catalog. In fact, different structures could arise based on application needs and it can dynamically evolve over time. The structure used and illustrated in Figure 2.3 is inspired by the one utilized at the UCSC Genome Table Browser[4].

The semantic catalog in Figure 2.3 is organized as a set of topics in a tree structure; the root node is the most general topic (i.e., the whole catalog) while leaf nodes represent the most specific topics (i.e., schema definitions). Topics are labeled and each one has an associated unique identifier, `TID`. This can be evaluated by traversing the tree starting from the root, the root topic has `TID=0`. The depicted dark route in the figure highlights the path traversed to evaluate the `TID` for the *"Expression/Regulation"* topic (`TID=0.2.2.1.2`).

The tree structure facilitates an XML representation of the catalog with the associated query mechanisms, and a synchronization mechanism based on WSRF[5] notification. How-

[4]http://genome.ucsc.edu/cgi-bin/hgTables
[5]http://www.globus.org/wsrf/

ever, other structures could be used if the appropriate query and synchronization mechanisms are provided.

Topics play an important role in the devised notification mechanism (discussed later in the *Synchronization Service*). Every node in the proposed architecture defines its own set of "topics of interest". Individual topics are only replicated on nodes that identify them as "topics of interest". That is, the system does not replicate the entire catalog on all nodes. This technique has three advantages. 1) It limits the size of the semantic catalog on specific nodes. 2) Topics can help large systems define specialized clusters of nodes. 3) It helps preserve autonomy of nodes (each node has full control on the contents of it's local catalog). Note that, if all nodes identify the entire catalog (`TID=0`) as a topic of interest, then the full catalog will be replicated on all nodes. Figure 2.3 illustrates the distributed data repository and depicts how semantic catalog topics can link to data resources stored there. Leaf topics represent schema definitions, and they can point (using URIs[6]) to one or more data resources. A basic URI is needed to point to the schema definition (XSD) file, and additional URIs can point to one or more instances of this definition (i.e. XML files). In addition, pointers to schema mapping files that directly relate pairs of schemas can be specified. Such mappings, created by the *Schema Mapping Creation service* described later, specify the exact semantics of data conversion from one schema to another, and can be subsequently used by other data services.

Finally, an additional point can be noted regarding the semantic catalog. Two kinds of schema topics are identified in the semantic catalog illustrated in Figure 2.3, specifically the *schema* and the *concepts schema*. *Schema* follows the conventional definition while *concept schema* is created by mapping schema elements to concept unique identifiers (CUIs) from a specific conceptual model (e.g. using the UMLSKS[7] Metathesaurus). For example, a schema element representing *gene identifier* can be named *geneID* in one schema and *gID* in another, however both will be mapped to the same UMLSKS CUI: *C1706509*. Typ-

[6]Uniform Resource Identifiers

[7]Unified Medical Language System Knowledge Source Server, http://umlsks.nlm.nih.gov

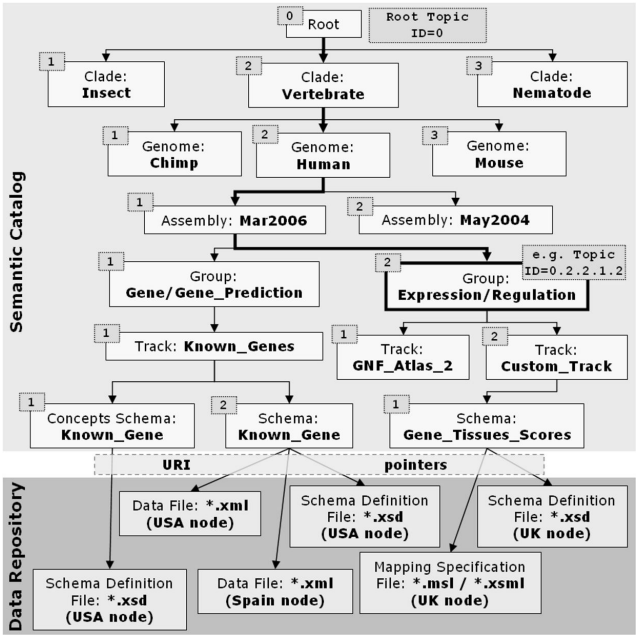

Figure 2.3: Semantic catalog structure and relation with data repository elements

ically, the conventional schema and its corresponding concepts schema exhibit the same structure. Mutual correspondences between conventional and concept schemas are stored in the semantic catalog. Concept schemas can be useful for deriving relations among different schema definitions (e.g. to discover if a given schema is equivalent, a superset, or a subset of an existing one).

Repository Service and the Data Repository: Currently, this service is implemented on top of the Apache Commons Virtual File System (VFS)[8]. VFS provides APIs for accessing different file systems and presents a unified view of files from different sources (local disk, remote ftp, or http servers).

The repository service is responsible for storing and extracting all raw data (via the VFS). It also notifies the synchronization service if there are any changes to the local repository. In the current implementation, only a pure XML data repository is supported. Other data representations can be supported only if the data can be exported as XML.

Synchronization Service: The synchronization service keeps the local semantic cat-

[8]http://jakarta.apache.org/commons/vfs/

alog entries synchronized with other nodes. When a node is added, it has the option of subscribing itself to various topics. Whenever a change occurs in the semantic catalog (of one node) affecting a certain topic, nodes that are subscribing to that topic receive notifications with the update. This service can use the WSRF notification mechanism provided by the Globus Toolkit.

MR-LEGOS Service: This is a data processing service, that can receive definitions of bioinformatics data processing tasks, and can submit them to a MapReduce cluster for processing. MR-LEGOS is a tool for composing MapReduce constructs from simpler components, namely, "Maplets", "Reducelets" and optionally "Combinelets". Using MR-LEGOS, complex bioinformatics problems can be defined in an easier and more efficient way. Standard bioinformatics processing components (i.e., bioinformatics-specific Maplets and Reducelets) can be shared and reused by groups of scientists in a highly collaborative way. The details of MapReduce and the MR-LEGOS model will be addressed in Chapter 4.5.6. The MR-LEGOS service also communicates with the repository service, so it can retrieve input data files for processing, and finally store the result into the repository.

Data Services Suite: This component provides a number of web services that allow the creation of schema mappings and operations over those schema mappings. We have selected Clio's [63] schema mapping components, and wrapped them into web services. The suite provides the following data services:

- **Schema Mapping Creation**: Given a source and a target schema, and a set of "correspondences" between source and target schema elements, this service creates a "mapping" from the source to the target schema. This mapping consists of a set of declarative constraints that dictate what the target instance should be, given a source instance. The mapping creation algorithm takes into account the schema constraints (e.g., foreign key constraints, type constraints) as well as the correspondences [109].

- **Query Generation**: Given a mapping (produced by the Schema Mapping Creation service), this service produces an XQuery, XSLT, or a SQL/XML query that imple-

ments the transformation implied by the mapping [63]. The query and its associated mapping are stored in the semantic catalog (for further reuse).

- **Query Execution**: For convenience, we also have a service that executes the queries generated by the previous service. Given a query script and a set of input XML documents (instances of the source XML schema used in the mapping, for example), the service executes the query and returns the resulting XML document.

- **XML Transformation**: We also provide a service that allows the direct and scalable execution of the mapping, as opposed to simply executing the query that implements it. Based on the technology detailed in [75], this service takes as input a mapping and the source XML instances and returns the target XML instance that is implied by the mapping. As opposed to the Query Generation or Execution services, this service neither produces nor executes a query; rather, this service uses a Java based engine that optimally executes the mapping.

- **Query Rewrite**: An interesting application of mappings is the ability to rewrite (or translate) target-side queries into queries that work on the source-side. This is useful, for example, if the target side schemas are virtual and the actual data resides on the source side. We use the query rewriting techniques detailed in [148] to implement this service. Given a schema mapping and an XQuery over a target schema instance, this service returns a rewritten XQuery over the source schemas in the mapping. When the rewritten XQuery is executed against the source document instance, the resulting document has the same structure as the intended output of the original target XQuery.

- **Schema Integration**: Given a number of mappings between several schemas, this service attempts to create an "integrated" schema that captures the unified concepts of the schemas that are related by the mapping[9]. Schema integration is a core com-

[9]The service currently provides a ranked list of possible integrated schemas for the user to choose from.

ponent of the presented thesis, and it will be independently addressed in more details in Chapter 3.

2.3 System Usage

We present two example usage scenarios for the BioFederator, the first example describes a collaboration scenario where geographically-distributed groups of scientists, working on the same project, are using the BioFederator in coordinating and managing the data and metadata required for their project. The second example describes a usage scenario that involves using the BioFederator in accessing the metadata for a bioinformatics problem, and then conducting distributed data processing using MR-LEGOS.

2.3.1 A Collaboration Scenario

This is an example collaboration scenario that can be achieved using the BioFederator. The usage scenario involve three collaborating groups of scientists. Assume the groups are associated with the three data sources shown in Figure 2.1 and located in the USA, Spain, and Mexico respectively. The USA group is conducting experiments related to identifying known genes, their chromosomal positions, etc. The team from Spain is doing experiments on gene expression levels in different tissues, while the team from Mexico is concerned about identifying transcription factors binding sites for different genes and the associated transcription factors. Further, assume there is a fourth team in the UK that will do the analysis of the collected data; their role is to collect and interpret data from different teams and to discover new knowledge from the experiments.

- **Initialization:** Each site in the study independently sets up and configures its own node. The configuration process includes the installation of any required software and the deployment of required services. As nodes join the system, their coordinator

services update lists of registered nodes, deployed services (both local and remote), and subscribe to the "topics of interest".

- **Data Storage:** Assume the USA group has conducted some experiments and collected data that needs to be deposited into the system. The first step is to define a suitable schema to represent this data, if one is not yet available. The next step is to prepare the actual data instance conforming to the designed schema and choose the associated parent topic (the topic should be defined among the hierarchy of topics in the *semantic catalog*). Optionally, mapping from schema elements to CUI, concept unique identifiers, can be provided to help construct the corresponding *concepts schema*. Having the schemas and actual data, the USA team connects to their node via an application server and starts uploading their data. The *coordinator service* delegates the *repository service* which handles the storage process (via VFS) and notifies the *synchronization service* to update the local *semantic catalog*. Figure 2.3 shows the uploaded schema *Known_Gene* topic pointing to the schema definition file and actual instances on the USA node. The *synchronization service* also notifies remote synchronization services, which are subscribed to the topic associated with the newly uploaded data. Note that the *coordinator service* may decide to store data on multiple nodes based on quality of service requirements. Figure 2.3 shows an additional instance saved on the Spain node.

- **Schema Mapping:** Assume now that the teams in Spain and Mexico have also uploaded their data. Thus, all three data sources have been loaded with data, which is hosted on different nodes. Now the analysis team, i.e. the UK team, can start interpreting and analyzing the data deposited by the other three groups. However, they are facing the problem of merging and integrating the data. Also, to efficiently analyze the data, they would like to organize it according to a specific structure. Therefore, the UK team constructs a new schema that captures the required data organization

(the target schema in Figure 2.1). The BioFederator includes a powerful schema mapping component that can create the mappings from the source schemas into the new target schema. The UK team connects and downloads (via Java Web Start) an application that allows the construction of Clio-based mappings. Source and target schemas are loaded into the tool which shows their structure as a tree of schema elements (very similar to how they are presented in Figure 2.1). Value mappings are entered by drawing lines from source schema elements to target schema elements. The *schema mapping service* processes the mapping specification and passes it to the *repository service* for storage. The *synchronization service* updates the local *semantic catalog* and notifies remote nodes about the new mapping. Figure 2.3 shows the target schema "*Gene_Tissues_scores*" pointing to both the schema definition file and the mappings file on the repository (UK node). When a decision is taken to construct a materialized instance of the target schema, the mappings specifications file is read and the source schemas TIDs are extracted.

- **Query Processing:** After constructing the new, federated target schema in the previous step, various analysis groups can start accessing and querying it. Different query processing scenarios could arise based on the location of data (which can be either locally or remotely stored), and whether the query is against a materialized version of the data or not. If the data is not materialized then either a materialization or a query distribution decision could be made by the *coordinator service*. Criteria for such decision can be based on the frequency of the queries against the data sources, and it can be locally materialized if the number of queries received exceeds a specific threshold. For instance, imagine the UK-team is trying to answer the following query using the federated target schema:

Find a list of **genes names** *and* **their chromosomal locations** *having an* **expression level** $\geq e$ *in both* **heart** *and* **liver** *and they are* **regulated by the same set of transcription factors**.

The above query is written against the federated target schema. However, note that the UK node does not have any data associated with this federated schema; all data resides at the other nodes (a *Global-Local-As-View* (GLAV) scenario [54]). One alternative could be data materialization (i.e., creating an instance of the target schema) using *query generation* and *execution services*, and then executing the query against this instance. Another alternative is using the *query rewrite service* as discussed before.

2.3.2 A Data Query and Processing Scenario

We present a data query and processing scenario that can be achieved using the BioFederator. The usage scenario involves two collaborating groups of scientists. The first group is conducting micro-array studies to calculate the gene expression profiles across different types of human tissues. The second group uses the data deposited by the first group to search for specific DNA patterns in house-keeping and tissue-specific genes, their objective is to study the characteristics of the DNA sequences for different genes and correlate such characteristics with the types of genes and their expression profile.

- **Initialization:** Each site in the study independently sets up and configures its own node. The configuration process includes the installation of any required software and the deployment of required services. As nodes join the system, their coordinator services update lists of registered nodes, deployed services (both local and remote), and subscribe to the "topics of interest".

- **Data Storage:** Assume the first group has conducted some experiments and collected data that needs to be deposited into the system. The first step is to define a

suitable schema to represent this data, if one is not yet available. The next step is to prepare the actual data instance conforming to the designed schema and choose the associated parent topic (the topic should be defined among the hierarchy of topics in the *semantic catalog*). Optionally, mapping from schema elements to CUI, concept unique identifiers, can be provided to help construct the corresponding *concepts schema*. Having the schemas and actual data, the first team connects to their node via an application server and starts uploading their data. The *coordinator service* delegates the *repository service* which handles the storage process (via VFS) and notifies the *synchronization service* to update the local *semantic catalog*. The *synchronization service* also notifies remote synchronization services, which are subscribed to the topic associated with the newly uploaded data. Since, the second team had subscribed to this topic, their local *semantic catalog* will be updated with the newly deposited data.

- **Query Processing:** The second group can now start accessing and querying the deposited data. For instance, assume the second group is trying to answer the following query:

  > *Find a list of* **genes names** *and* **associated DNA sequences file locations** *where* **gene_type =** $house - keeping$.

  The above query is written against the schema created by the first group. The query will be executed and the results can be retrieved by the first group.

- **Data Processing:** After retrieving the results from the previous step, The second group wants now to analyze the DNA sequences associated with every gene in the result set, and to search for specific DNA patterns. However, the result set contains thousands of genes, and retrieving and processing these amounts of data can be time consuming. The second group decided to use MR-LEGOS to address this data processing problem. The team wrote an MR-LEGOS job definition, that will read

and process these gene sequences in parallel. The MR-LEGOS job definition used a bioinformatics sequence localization Maplet, that basically reads a DNA sequence file and calculate and write the locations of the patterns of interest. This Maplet was written by a different research group, but the second group was able to reuse this same Maplet. The group submits this job definition via the BioFederator *application server*, the request is sent to the *coordinator service* of the local node, that forwards it to the MR-LEGOS service.

Chapter 3

Schema Integration Approach

The need for a unified representation of the data to simplify the access to heterogeneous data sources leads to the topic of schema integration. The input to this problem is a set of source schemas together with attribute correspondences relating them. The output is a consolidated target schema that constitutes a nonredundant unified representation of all the data.

Schema integration has been an active research field for a long period of time and continues to be a challenge in practice [22, 26, 96, 128, 111, 33]. This problem lies at the core of many metadata applications, such as view integration, mediated schema creation, and ontology merging. The spectrum of applications extends to web-service integration, mashups and distributed web architectures [115].

In this thesis, we build on a number of previous approaches. The study in [33, 34] proposed a schema integration framework that is based on enumerating and exploring multiple candidate integrated schemas based on considering all possible choices of merging or not merging pairs of concepts that have matching attributes. This set of choices gives rise to a *space of candidate integrated schemas*. This method goes on to explore this space of integrated schemas in an *interactive* way, where user constraints are used to direct the exploration of the solution space. In [55] a related approach utilizing top-k schema map-

pings for managing the uncertainty in schema matching was proposed. The proposed top-k algorithms presented in this study has some limitations that we address in this thesis.

In this thesis, we study a different exploration strategy that is based on *ranking* the candidate schemas. As in [33, 34], we use graphs of *concepts* with has relationships to represent, in a logical way, XML or relational schemas. A concept intuitively represents one category of data (an entity type) that can exist according to that schema (e.g., an "employee", a "dependent", an "address", etc.).

Our contributions are as follows. First, we develop a distance measure, which we call *concept distance (CD)*, that quantifies the similarity and coverage of concepts in different schemas, our distance measure includes a *directed* component that allows us to quantify the degree of coverage of one concept by another; and hence, say that one concept is likely to be an extension or a sub-concept of a second concept.

As a consequence of developing a concept distance, we are then able to *rank* the candidate integrated schemas by giving more weight to schemas that combine the concepts that have high similarity or coverage. The number of candidate schemas can be quite large, in general, thus prohibiting the exploration of the whole solution space. Accordingly, we devise a $top\ k$ enumeration algorithm for generating the top candidate schemas without going through the entire space. The complexity of the algorithm is analyzed showing enhancement over previous approaches [55, 33, 34].

We note that, schema matching techniques for measuring $undirected$ similarities between complex schema elements have been proposed [118, 93, 25]. However, in this thesis, the emphasis is on the directed nature of our distance measure (CD) and how it is applied in the context of schema integration. Furthermore, our method is complementary to schema matching, since it maintains and uses the outcomes of schema matching in the ranking of integrated schemas.

Schema integration seeks to derive a unified representation of the data, used to simplify the access to heterogeneous data sources. The input to integration is a set of source

schemas that relate to each other through correspondences or constraints. The output is a consolidated target schema that constitutes a nonredundant unified representation of all the data.

Although today the process of integrating schemas is partially automated, it is still labor-intensive. In order to reduce the amount of manual intervention that is required from users, we need to modify or avoid parts of the integration process that unnecessarily increase the load on users. Let us follow the steps that generally need to take place while combining two input schemas.

First, the input schemas are run through one or more schema matching algorithms [118, 93, 25] that return *correspondences* between the elements of the schemas. Such correspondences typically have weights reflecting the confidence of the matchers that the two elements are similar or have overlapping semantics. Moreover, the weights in each correspondence direction can be different; thus, an element A can be similar to (or covered by) an element B with weight w, while the element B is similar to (or covered by) element A with weight w', where w and w' are not necessarily equal. We say in such situation that there are two *directed and weighted correspondences* between elements A and B. In a second step of schema integration, all the directed correspondences[10] between two elements are merged into one undirected correspondence with one aggregated weight. Correspondences for which the aggregated weight is above a threshold are kept, while the rest are discarded. Moreover, after the above pruning step, the weights of the remaining set of correspondences are typically themselves discarded. In the third step, several alternatives for combining the input schemas are available, based on the surviving correspondences, and schema integration tools provide interactive means for the users to select a desired integrated schema.

Consider the simple example in Figure 1.1. The two input schemas describe the structure of two elements: *householder* and *member*. These schemas are run through one or

[10]There may be more than two, with multiple schema matchers.

more matching algorithms. For simplicity, assume in this example that atomic elements that match are assigned correspondences weighted with a similarity of 1, in both directions. Correspondences that have weight 0 are not shown. The schema matching algorithm then calculates the overall similarities between the non-atomic elements, by aggregating in some way the similarities of the sub-elements. In this example, all the sub-elements of *member* match the sub-elements of *householder*, and therefore *member* is covered entirely by *householder*. Thus, we can conclude that there is a directed correspondence *member* $\rightarrow$ *householder* with the similarity/weight of 1. On the other hand, the element *householder* contains some sub-elements that are unique, and as a result the similarity of the directed correspondence *householder* $\rightarrow$ *member* is less than 1.[11] One can see that the weights of the derived correspondences give valuable hints about the coverage of one element by another. Such information is more refined than just saying whether two elements match or not. In particular, it can suggest that one element is likely to be an extension or represents a sub-concept of the other element.

Let us revisit the second step in schema integration, where the results of the matching algorithm are transformed into the undirected correspondences used for the generation of the integrated schema. There are several ways to combine the elements of the two schemas, but in this simplified example there are three natural choices: (1) merge the two root elements *householder* and *member* into one, (2) introduce an extension relationship (or reference) that will say that *householder* is an extension of *member*, or (3) do not combine the two elements at all and just take their union. This is a simple example; in reality schemas are larger and have more complex relationships among their elements. Thus, the space of *possible* candidate schemas that can result from combining the input schemas can be quite large.

The third step in the schema integration process is then concerned with identifying the "best" integrated schema among these alternatives. In most methods, the alternatives are

[11] Section 3.3 will show one method for calculating the similarity measure for complex elements

not created explicitly in the system, and the user must "drive" the generation of *one* integrated structure. A good example of such system is described in [111]; in their method, the expert provides a "template" of the integrated schema (also called a mapping model) that effectively specifies the structure of the merged schema and provides the basis for accumulating all the attributes and the relationships from the input schemas. A different kind of method, that is based on the explicit identification of the alternative schema structures, is described in [34]. In their system, a user can systematically explore the alternatives and narrow down, in an interactive way, the desired integrated schema. The advantage of such method is that it is based on a systematic enumeration of the design choices and provides more information to a user. The disadvantage is that the user can be exposed to many "unlikely" schemas and has to explicitly rule out the unlikely choices. Figuring out a way of directing user's attention to the more challenging integration choices was left as an open problem.

3.1 Overview of Our Approach

In this thesis we address the above shortcomings in the schema integration process as follows. (See also Figure 1.2.) First, we keep and exploit all the information generated by the matching algorithms. In particular, we make use of both the direction and the weights associated with the correspondences between elements. This information enables us to define more refined relationships on the integrated schema. More importantly, this information enables us to *rank* the integrated schemas, by giving higher priority to schemas that combine the concepts with higher similarity or coverage. We are then able to devise a polynomial-time, efficient algorithm that generates the top-k integrated schemas according to the above heuristic.

As a consequence, the resulting system can avoid the generation of many unlikely schemas and can thus minimize the user effort. As our user experiments show, the top-k

schemas that we generate will automatically make the correct decisions in the "easy" (and frequent) cases (i.e., concepts that are highly similar will be combined, while concepts that are highly dissimilar will not be combined). Thus, the users can focus their attention on the "difficult" cases while letting the system do the "tedious" part. In fact, we show in Section 3.6.2 that the top-k algorithm can be effectively combined with the interactive system of [34] as follows. The system first generates the (unconstrained) top-k schemas according to the initial input (schemas and correspondences). The user then inspects some of these schema and adds one or more *constraints* in the sense of [34], restricting the space of possible schemas. These constraints enforce decisions to be made on the "difficult" cases. Next, the system reacts by generating the top-k schemas that satisfy the constraints. The process continues, with the user possibly adding some more constraints, until a final schema is obtained.

The top-k generation algorithm itself is non-trivial and we include the proof of its correctness. Furthermore, we show that the exact complexity of the algorithm is, in the worst case, $O(m_1 m_2 \log(m_1 m_2) + k^2)$, where m_1 and m_2 are the numbers of concepts in the two schemas, respectively. We then show experimentally that the algorithm performs well in practice, on real schemas.

Our method uses an extension of the framework of [34] for schema enumeration. As in [34], we use graphs of *concepts* with has relationships to represent, at a higher-level of abstraction, XML and relational schemas. A concept of a schema is a relation name with an associated set of attributes and, intuitively, represents one category of data (an entity type) that can exist according to that schema (e.g., an "employee", a "department", an "address"). Concepts in a schema may have references to other concepts in the schema and these references are captured by has edges (e.g., "employee" contains a has edge to "department").

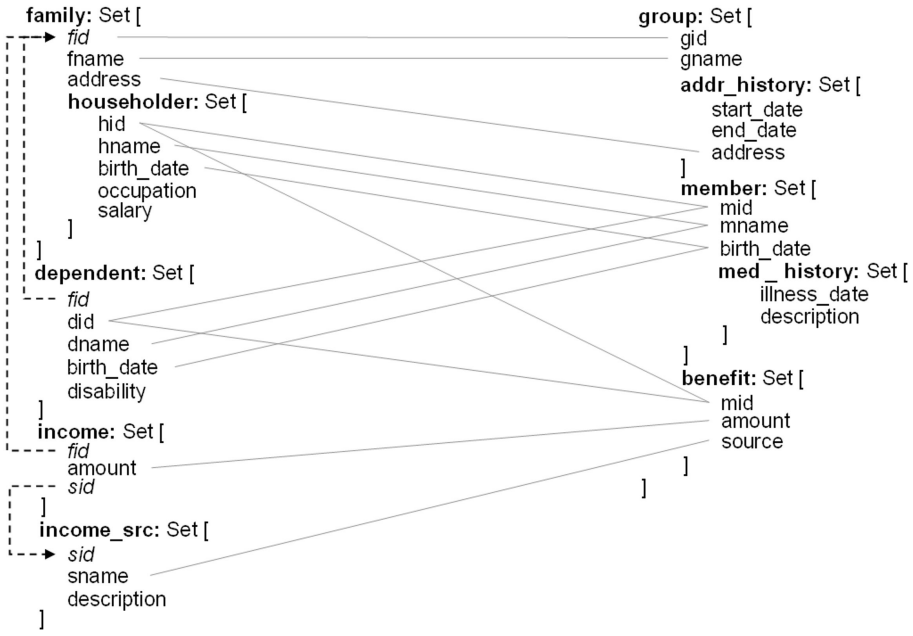

Figure 3.1: Two input source schemas and their attribute correspondences.

3.2 Model Formalization

Schemas and Correspondences Consider the two source schemas S_1 and S_2 shown in Figure 3.1. The schemas are shown using a nested relational model [110] which can be used as a common representation for both relational and XML schemas, in addition to other hierarchical set-oriented data formats. This model is based on sets and records that can be arbitrary nested.

The first schema represents families with their householders, dependents and incomes, as well as the sources for such incomes. The top-level $family$ element represents a *set* of family records, each with three atomic components and one nested set, $householder$, representing the householders for each family. The dotted arrows in schema S_1 represent foreign key constraints: an income has references to both a family and an income source, while a dependent has a reference to a family. The second schema is a variation of the first one, where corresponding to $family$, at the top level, we have a set of $group$ records. Each $group$ record includes nested sets of $addr\_history$, $member$ and $benefit$. As the example illustrates, in general, the source schemas can overlap and also each source schema can

have information that is not present in the other (e.g., $salary$, $disability$ in the first schema, $addr\_history$, $med\_history$ in the second schema).

Figure 3.1 also shows correspondences (lines) between attributes of the two schemas. The correspondences signify "matching" or "similar" attributes (i.e., that carry $similar$ data) in the two schemas. They can be manually specified or discovered through schema matching techniques. Such techniques typically examine attribute types and properties, and may also, involve calculating similarities based on available data instances for corresponding attributes. Only correspondences between attributes (i.e., atomic components) are considered, since attributes carry the actual data. Note that there can be attributes with no correspondences and also attributes with multiple correspondences.

In general, some correspondences can have "stronger" similarity than others. Our framework and, in particular, the calculation of distances between concepts that we shall describe later, is capable of incorporating weighted correspondences (with weights between 0 and 1). However, for simplicity, in this example and later we shall assume all correspondences have equal weights (of 1).

Given the source schemas and correspondences, our goal is to generate an integrated schema that captures the source schemas, by taking the union of all their features while avoiding redundancy (as captured by the correspondences). In general, there are many possible such schemas; in the following, we shall develop techniques for the ranked exploration of the candidate integrated schemas.

As in [33, 34], our integration method is based on a more logical representation of schemas that uses concept graphs.

Concept Graphs The objective of concept graphs is to abstract the physical organization of schemas into a more logical view. In the following, let us assume U is a universe of attributes. A *concept* is a relation name C associated with a subset of U (these are the attributes of C). A *concept graph* is a pair (V, has) where V is a set of concepts and has

is a set of directed edges between concepts. Whenever there is a has edge from A to B we say that A "has a" B or that A "extends" B or that A is a "sub-concept" of B.

Relationship Multiplicity Intuitively, the meaning behind a has edge from A to B is that every instance of concept A has a reference to exactly one instance of concept B. The role of the has edges is to express that certain concepts cannot exist without other concepts (i.e., they *extend* or *depend on* those concepts). Also, another way to view an edge A has B is that it represents a many-to-one relationship from A to B, i.e., there can be zero or more A instances with references to the same B instance. A many-to-many relationship between A and B can be represented using a new concept C with two has edges pointing to A and B, respectively. We note that, in general, there could be more than one has edge between two concepts and, moreover, the graph can have cycles. The above discussion clarifies how all possible relationship multiplicities can be represented using the proposed concept graph constructs. Accordingly, such multiplicity doesn't affect the behavior of our proposed integration approach since the inputs to the integration step are the concept graphs (as discussed in more details in Section 3.4).

Each schema, with its nesting and constraints, can be replaced by a concept graph. For example, two concept graphs are shown in Figure 3.2 corresponding to schemas S_1 and S_2. The non-dotted arrows that connect concepts (e.g., $householder$ to $family$ or $income$ to $income\_src$) represent has edges. For now, let us ignore the dotted lines connecting concepts across schemas.

To understand how these concept graphs relate to the schemas, consider the concept graph for S_1. There, $family$ and $income\_src$ are top level concepts, with no outgoing has edges; they represent standalone concepts that do not depend on anything else (according to S_1). In contrast, $householder$ has a reference to $family$, since a householder element cannot exist independently of a family according to the nesting in S_1. Similarly, $dependent$ has a reference to $family$, based on the fact that $dependent$ has a foreign key to $family$

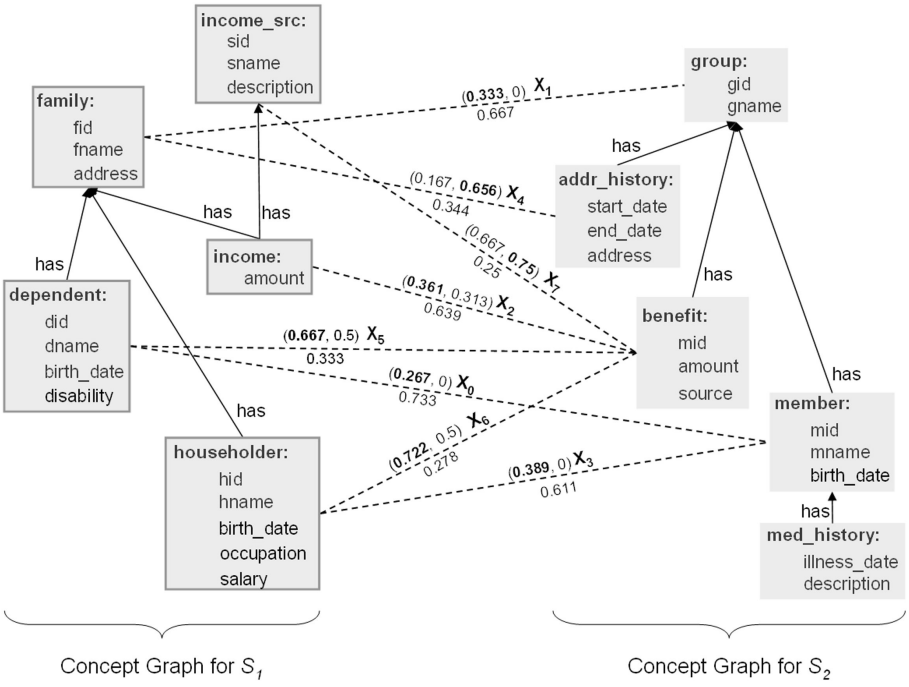

Figure 3.2: Concept graphs together with their distances and similarities.

in S_1. Moreover, $income$ has references to both $family$ and $income\_src$, since an income is associated (via foreign keys) with both a family and an income source in the schema S_1.

There is an immediate algorithm that constructs a concept graph from a schema, based on the nesting structure and based on the integrity constraints [34]. Specifically, a concept can be associated with each set element in the schema (e.g., family, householder, etc.). Furthermore, if the set element is nested under another set or has a foreign key into another set, then a corresponding has edge must be added. The study in [33] also provides techniques for translating concept graphs back into schemas. Accordingly, for the purposes of our subsequent steps, we can assume that the schemas are given as concept graphs and the schema integration problem is now a problem of combining such concept graphs into a unified graph. In general, our integration method can be applied on any input concept graphs and not only the ones constructed from schemas.

3.3 Concept Distance

In this section we introduce Concept Distance (CD), a distance measure for concepts and concept graphs. In essence, this is a variation of the well-known Hausdorff distance [100]. The CD measure is capable of quantifying the distance between concepts by incorporating both the attributes within concepts and the references to other concepts. We also emphasize the importance of both the undirected and directed versions of this measure, which will play key roles in our method.

3.3.1 Motivation

The body of schema matching literature contains numerous techniques for calculating similarities/distances between schema elements [118, 93, 25]. The emphasis in this study is not on the proposed similarity measure, but on its directed nature. A number of exiting techniques may be adapted to provide such asymmetry and hence, can be used in our approach.

The proposed distance measure has two main appealing characteristics that motivated its utilization. First, it can be simply defined and calculated, and second, it is a straightforward extension of a well-known, widely studied and accepted distance measure between sets, namely, Hausdorff distance.

A number of reasons motivated considering and studying such asymmetry in distance calculation. The study in [55] presented a refined categorization for $1 : n$ schema elements matchings. Such cardinality constraint may indicate that a single attribute in one schema is replicated to several attributes in another schema (e.g., $email$ in one schema vs. $email$ and $confirmEmail$ in the other schema). Alternatively, such constraint may indicate that an attribute in one schema is decomposed into several attributes in another schema (e.g., $name$ in one schema is decomposed into $firstName$ and $lastName$ in another schema). Using undirected similarity measures, there is no way of representing and capturing such

alternatives. Introducing asymmetry and the notion of coverage in both directions provide a possible way of capturing such information.

To exemplify the use of asymmetry in the context of schema integration, assume we have two relations, $Employee$ (id, $name$, $salary$) and $Manager$ (id, $name$, $salary$, $bonus$, $dept$, $office$, $secretary$). A undirected distance measure will calculate a single value as a measure of the similarity between the two relations. A schema integration algorithm will make a decision based on this single value, and hence, decide either to merge those relations or to keep them separate. Note that, both decisions are not accurate and a possible more realistic decision is obvious, namely, a partial merge, where $Manager$ becomes an extension for $Employee$. We'll later show that our integration framework is capable of making such decisions. This process has appealing characteristics from a database normalization perspective. More details about this will be discussed later.

The directed nature of a similarly measure has favorable characteristics in a number of application scenarios, as can be seen from the above discussion. And, if required, the directed components can be combined into a single undirected measure. Hausdorff distance provides its directed components in addition to a number of well studied undirected measures. We utilize and extend those measures in the following.

3.3.2 Distance Between Object Sets

We start by reviewing the basic notions related to the Hausdorff distance. We shall use $d(a, b)$ to denote the distance between two objects a and b; correspondingly, $s(a, b) = 1 - d(a, b)$ represents the similarity between a and b. Both measures are directed, i.e., $d(a, b)$ is not necessarily equal to $d(b, a)$, and vary in the range [0,1]. The distance between an object a and a *set* of objects $B = \{b_1, ..., b_{N_B}\}$ is defined as:

$$d(a, B) = \min_{b \in B} d(a, b) \tag{3.1}$$

There are several ways to define the directed distance between two object sets $A = \{a_1, ..., a_{N_A}\}$ and $B = \{b_1, ..., b_{N_B}\}$ as shown in [73]. The study in [48] proposed the following directed modified Hausdorff distance:

$$d(A, B) = \frac{1}{N_A} \sum_{a \in A} d(a, B) \tag{3.2}$$

The complement of the above formula (i.e., the similarity $s(A, B) = 1 - d(A, B)$) represents a measure that quantifies the degree to which the set A is covered by the set B. In other words, the measure indicates the degree to which A can be considered a "subset" of B. This similarity measure varies in the range $[0, 1]$, where 1 denotes complete "inclusion".

3.3.3 Distance Between Concepts

We now use a variation of the previously discussed notion of distance between sets of objects to define distances between concepts. For the purposes of this definition, a concept can be thought of as a set of attributes. However, the definition will necessarily be recursive because it must take into account, for each concept, all its references. For this section, we assume that for each of the input concept graphs, the has edges do not form a cycle. We shall describe separately in Section 3.3.4 how we handle cycles.

Let $\alpha(X)$ denote the collection of $attributes$ in a concept X, and let $\beta(X)$ denote the collection of concepts to which X has direct references (i.e., has edges). We define the concept version of Equation (3.1) as:

$$\hat{d}(a, B) = \min \left(min_{b \in \alpha(B)} d(a, b), min_{B' \in \beta(B)} \hat{d}(a, B') \right) \tag{3.3}$$

and the concept version of Equation (3.2) as:

$$\hat{d}(A, B) = \frac{1}{N_\alpha + N_\beta} \left(\sum_{a \in \alpha(A)} \hat{d}(a, B) + \sum_{A' \in \beta(A)} \hat{d}(A', B) \right) \tag{3.4}$$

where N_α and N_β are the numbers of $attributes$ and, respectively, the number of outgoing

has edges of A, and where $d(a, b)$ is 0 if there is a correspondence between the attributes a and b, and 1 otherwise. Note that both of these definitions are recursive.

To better understand how this works, let us consider an example.

Example 1 *Consider the concept graphs shown in Figure 3.2. Let us calculate the distance between the $family$ concept and the $addr\_history$ concept.*

We have $\alpha(family) = \{fid, fname, address\}$ and, $\alpha(addr\_history) = \{start\_date, end\_date, address\}$. Furthermore, $\beta(family) = \emptyset$ and $\beta(addr\_history) = \{group\}$.

The directed distances between the two concepts are evaluated as follows:

$\hat{d}(family, addr\_history) = \frac{1}{3}(\hat{d}(fid, addr\_history) + \hat{d}(fname, addr\_history) + \hat{d}(address, addr\_history)) = \frac{1}{3}(\hat{d}(fid, group) + \hat{d}(fname, group) + d(address, address)) = \frac{1}{3}(d(fid, gid) + d(fname, gname) + 0) = \frac{1}{3}(0+0+0) = 0$

$\hat{d}(addr\_history, family) = \frac{1}{4}(\hat{d}(start\_date, family) + \hat{d}(end\_date, family) + \hat{d}(address, family) + \hat{d}(group, family)) = \frac{1}{4}(1 + 1 + 0 + \hat{d}(group, family)) = \frac{1}{4}(1 + 1 + 0 + \frac{1}{2}(\hat{d}(gid, family) + \hat{d}(gname, family))) = \frac{1}{4}(1 + 1 + 0 + \frac{1}{2}(0 + 0)) = \frac{1}{4}(1 + 1 + 0 + 0) = 0.5.$

The first calculation in the example implies that the $family$ concept is totally covered by the $addr\_history$ concept, since $\hat{d}(family, addr\_history) = 0$. In particular, the similarity of $family$ to $addr\_history$ is 1, according to the calculation. However, this result is not quite satisfactory, since most of the coverage of $family$ comes by using the attributes of the $group$ concept that $addr\_history$ refers to.

Intuitively, when we compare $family$ with $addr\_history$, we would like to decrease the resulting similarity by taking into account the fact that the attributes fid and $fname$ in $family$ are covered via a referenced concept and not directly by $addr\_history$.

To achieve this intended behavior, we modify the above computation as follows. Assume that we need to compute the distance between concepts A and B. Then, we shall keep track of the number δ_A of has edges that we follow from A (with each recursive call to Equation (3.4)). Similarly, we shall keep track of the number δ_B of has edges that we follow from B (with each recursive call to Equation (3.3)). Then, whenever we reach the fixed point in the recursion where we need the distance $d(a, b)$ between attributes a and b (see Equation (3.3)) we use a scaling factor κ to alter this distance, based on the *difference* between δ_A and δ_B. More precisely, we take $\kappa = \frac{\alpha}{\alpha+\delta}$, where $\delta = |\delta_A - \delta_B|$ and where α is an application dependent parameter. We then alter the similarity between a and b by using

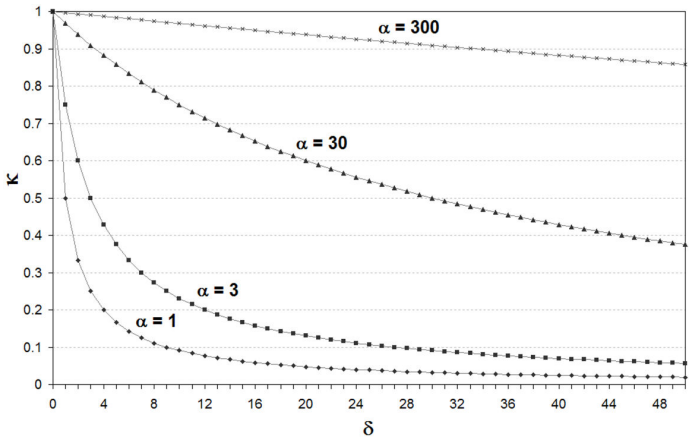

Figure 3.3: Behavior of κ using different α values

a weighted similarity formula $\hat{s}(a, b) = \kappa \times s(a, b)$. Accordingly, the weighted distance between a and b is $\hat{d}(a, b) = 1 - \hat{s}(a, b)$. This number is then used in place of $d(a, b)$ in Equation (3.3).

The intuition behind κ is that we want to penalize the similarity between attributes only when the attributes reside in concepts that have *different* depths relative to the original concepts A and B that are being compared. This guarantees in particular that the distance between two identical concept graphs will always be 0. For our example, when we calculate the distance between $family$ and $addr\_history$ and reach the point of comparing fid in family with gid in $group$, the depth difference is 1; therefore κ will have a value that is strictly smaller than 1 (the exact value depends on the parameter α).

The behavior of κ against δ with different α values is depicted in Figure 3.3. It can be seen that as α increases the penalty based on depth distances decreases, and vice versa.

Using the updated formula, and taking α to be 3, we recalculate the distance between $family$ and $addr\_history$ in our example and obtain $\hat{d}(family, addr\_history) = 0.167$. A higher value for α would give a smaller distance.

In the later part of the study, we shall further need to combine these directed measures ($\hat{d}$ and $\hat{s}$) into undirected measures. The following formula can be used to produce a undirected distance measure:

$$\hat{D}(A,B) = \max(\hat{d}(A,B), \hat{d}(B,A)) \tag{3.5}$$

We then define the undirected similarity measure between two concepts A and B as:

$$\hat{S}(A,B) = 1 - \hat{D}(A,B) \tag{3.6}$$

The study in [48] analyzed various ways of combining directed distances into a undirected distance and reported better object matching performance when taking the maximum of the directed distances as in Equation (3.5) (as opposed to taking the minimum, the average or the weighted average). In the context of schema integration, we shall utilize the undirected distance to define the cost of combining two concepts.

3.3.4 Distance Calculation

Here we present the procedure for calculating the complete set of distances between concept graphs based on the previously defined CD measures. Assume we have n input schemas (S_1,...,S_n), each represented as a concept graph, and we are given a set of correspondences that relate pairs of attributes across schemas.

To calculate the complete set of distances, we iteratively calculate the distance between each pair of concepts across input graphs. The calculated distances (using $\alpha = 3$) are shown in Figure 3.2 using the dotted lines connecting concepts across schemas. From now on we shall use the term *distance_edges* for the edges that connect concepts in different schemas. For each such *distance_edge* we show the two directed distances right above the edge and the undirected similarity right below the edge. For example, the two distances $\hat{d}(family, group) = 0.333$ and $\hat{d}(group, family) = 0$ are shown as a pair above the *distance_edge* between $family$ and $group$. Additionally, the maximum of the directed distances is highlighted using a bold face font (this is the undirected distance $\hat{D}(A,B)$ using Equation (3.5)).

Table 3.1: List of distances between all concepts in S_1 and S_2

	group	***member***	***benefit***	***med_history***	***addr_history***
family	(0.333,0)	(0.5,0.906)	(0.5,0.906)	(0.7,0.981)	(0.167,0.656)
householder	(0.958,0.25)	(0.389,0)	(0.722,0.5)	(0.542,0.938)	(0.889,0.563)
dependent	(0.95,0.25)	(0.267,0)	(0.667,0.5)	(0.45,0.938)	(0.867,0.563)
income	(0.917,0.25)	(0.778,0.75)	(0.361,0.313)	(0.833,0.969)	(0.778,0.562)
income_src	(1,1)	(1,1)	(0.667,0.75)	(1,1)	(1,1)

The distances between each pair of concepts across the input schemas are calculated. Accordingly, there is a *distance_edge* connecting each concept in S_1 to each other concept in S_2. Table 3.1 shows this whole collection (using $\alpha = 3$). However, for illustration simplicity, Figure 3.2 only shows a subset of those edges, specifically it shows the *distance_edges* connecting a pair of concepts A and B if there is at least one attribute correspondence between attribute a and b, where $a \in \alpha(A)$, $b \in \alpha(B)$.

As an example, the calculated measures show that *householder* matches better (with higher similarity) with $member$ than with $benefit$. This is expected and it is something we shall take advantage of to rank the integrated schemas. In particular, we shall give higher weight to the schemas that combine *householder* with $member$ than to the ones that combine *householder* with $benefit$.

Handling Cycles Recall that our concept distances (Equations (3.4) and (3.3)) are recursive. Thus, cycles in the input concept graphs can cause problems for the computation of concept distances. Cycles arise naturally in practice (e.g., an employee has a department, and a department has an employee which is the manager). To handle cycles in our implementation, we incorporate a cycle detection technique, which keeps track of the visited concepts during recursion. When a cycle is detected, we loop in the cycle until the difference between two successive calculations $(\Delta\hat{d})$ becomes smaller than a considerably very low value ϵ. From the recursive nature of the formulas there is a guarantee that $\Delta\hat{d}$ converges (i.e., $\forall iter(\Delta_{iter}\hat{d} > \Delta_{iter+1}\hat{d})$), but since there is no guarantee of the rate of the convergence, for performance issues we break the cycle after a predefined number of iterations $iter_{max}$.

3.4 Overview of Schema Integration Method

This section presents an approach for constructing the integrated schema using the previously discussed directed distances. Figure 3.4 outlines the main steps, the n input schemas are translated into concept graphs, then the directed distances between all pairs of concepts across the graphs are calculated using simple attribute correspondences (as discussed in Section 3.3.4). Finally, using a parameter λ, the graphs are combined into *one* unified concept graph, which in turn can be translated into one integrated schema. The directed distances and the parameter λ play a key role in the concept graphs combination procedure as will be discussed in the following.

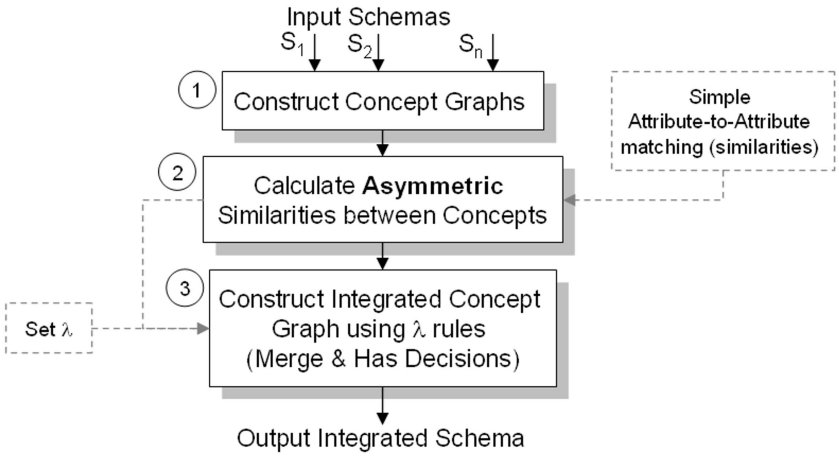

Figure 3.4: Schema integration workflow incorporating directed similarities

Based on directed distances, three different decisions for combining concepts are possible:

- $merge$: Two quite similar concepts can be merged into one, by taking the union of their attributes and the union of their has edges. At the same time, we avoid redundancy and collapse any corresponding (i.e. matching) attributes that may appear in this union into a *single* attribute in the merged concept.

- $no\ action$: Two highly dissimilar concepts will be left intact.

- Introducing a has edge: In some cases a $merge$ decision will be quite aggressive, and a $no\ action$ decision will be inaccurate. In these special cases, we propose adding a has edge connecting one of the concepts to the other. Specifically, assume that a concept A is covered relatively well by another concept B but B is not covered well by A (i.e., $\hat{d}(A, B)$ is low, but $\hat{d}(B, A)$ is high). This means that most attributes of A are "covered" by B, but the reverse is not true. We then make B an extension of A by adding a has edge from B to A and removing all the attributes from B that have corresponding attributes in A. Note that the new has edge did not exist previously in the input concept graphs but is something that is created by the schema integration procedure.

From a database normalization perspective, this enriched set of combination decisions helps minimize duplication of information and, in doing so, safeguards the database against certain types of logical or structural problems, namely data anomalies [36]. Previous schema integration approaches (e.g. [33, 34]) only considered $merge$ and $no\ action$ decisions, resulting in an integrated schema that may suffer from such problems. The use of directed distances together with the incorporation of partial merge decisions (i.e., has edges) help avoiding these problems.

3.4.1 Control of Combination Decisions

To control the combination process, a parameter λ is used to guide combination decisions. Assume two concepts A and B, where the directed distances between them are $\hat{d}(A, B)$ and $\hat{d}(B, A)$. If the distances in both directions are smaller than λ, then we choose to $merge$ the two concepts. If $\hat{d}(A, B) \leq \lambda$, but $\hat{d}(B, A) > \lambda$, then we choose to introduce a has edge, where B becomes an extension of A. Finally, if distances in both directions are larger than λ then a $no\ action$ decision will be the remaining option. The rules for $merge$ and has decisions are summarized in Table 3.2.

Figure 3.5 shows an example, where at the top we show concept $householder$ from

Table 3.2: $merge$ and has Decisions

Rule	Condition	Decision
1	$\hat{d}(A,B) \leq \lambda$ AND $\hat{d}(B,A) \leq \lambda$	$merge$
2	$\hat{d}(A,B) \leq \lambda$ AND $\hat{d}(B,A) > \lambda$	$B\ has\ A$
3	$\hat{d}(A,B) > \lambda$ AND $\hat{d}(B,A) \leq \lambda$	$A\ has\ B$
4	$\hat{d}(A,B) > \lambda$ AND $\hat{d}(B,A) > \lambda$	$no\ action$

schema S_1 and concept $member$ from schema S_2 together with the edge connecting them and their directed distances, $\hat{d}(householder, member) = 0.389$, while $\hat{d}(member, householder) = 0$. If we choose λ to be 0.6, a $merge$ decision will be taken and the result will be a single table for members and householders. This may not be fully correct because there may be family members who are not householders. If we take λ to be 0.3, a has edge will be introduced connecting $householder$ to $member$; this will result in a schema with two tables with a foreign key as shown in the figure, where, attributes in $householder$ having corresponding attributes in $member$ were removed. Note that a $no\ action$ decision would have also resulted in two tables but with redundant information.

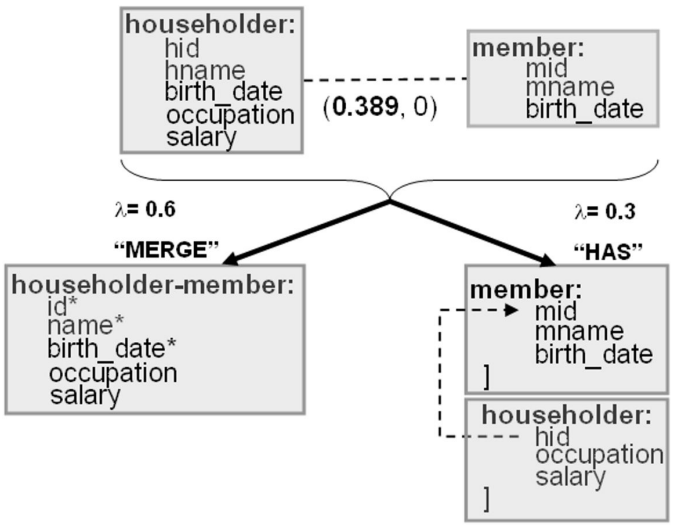

Figure 3.5: $merge$ and has decisions

3.4.2 Lambda Tuning

As seen from the rules for combining concepts (see Table 3.2), the choice of the parameter λ is critical, as it directly affects the generated schema. Figure 3.6 shows the combined

concept graph when $\lambda = 0.47$. Note that $family$ and $group$ are merged into one concept, as well as $income$ and $benefit$; moreover $dependent$, $member$ and $householder$ are also merged. Figure 3.7 manifests the effect of introducing a has edge from $householder$ to $member$ rather than merging $householder$ with $member$ when $\lambda = 0.37$. (Note that $dependent$ is merged with $member$, so the has edge from $householder$ goes into the merge of $member$ and $dependent$). Figures 3.6 and 3.7 also show the corresponding integrated schemas.

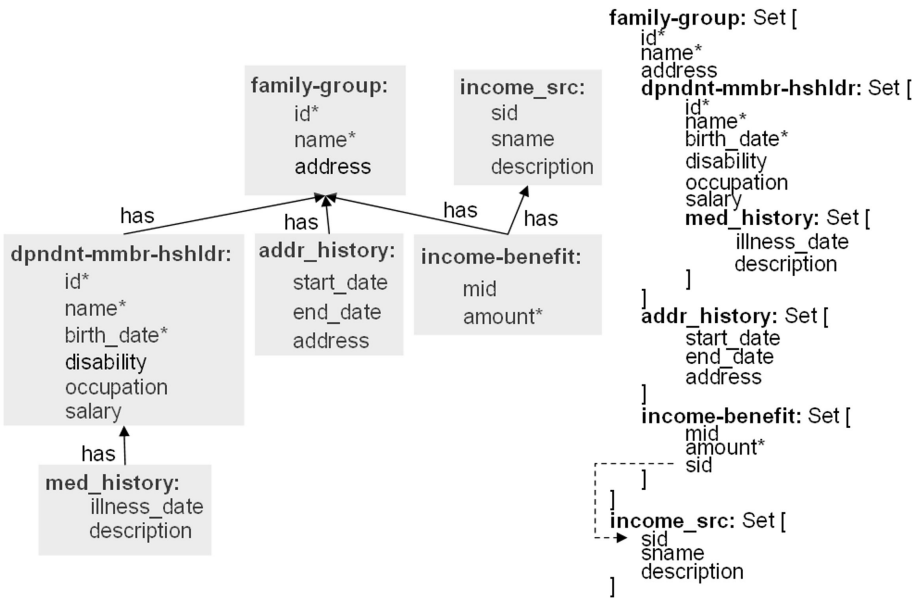

Figure 3.6: Combined concept graph and schema, for $\lambda = 0.47$

Note that, as λ increases, the number of *distance_edges* that are used (either for merge or to add a has edge) will also increase, thus resulting in combining more concepts and creating a more *"compact"* integrated schema. In the extreme case of $\lambda = 1$, all the *distance_edges* will be used and all concepts will be merged into one flat concept. As previously discussed, from database normalization perspectives, this unguided increase of λ is unsatisfactory. On the other hand, decreasing λ will limit the number of *distance_edges* that are used (either to merge or to add a has edge) to the ones connecting tightly related concepts, thus, resulting in a relatively *"fragmented"* integrated schema. In the extreme case, when $\lambda = 0$, only totally covered or identical concepts will be combined. Tuning

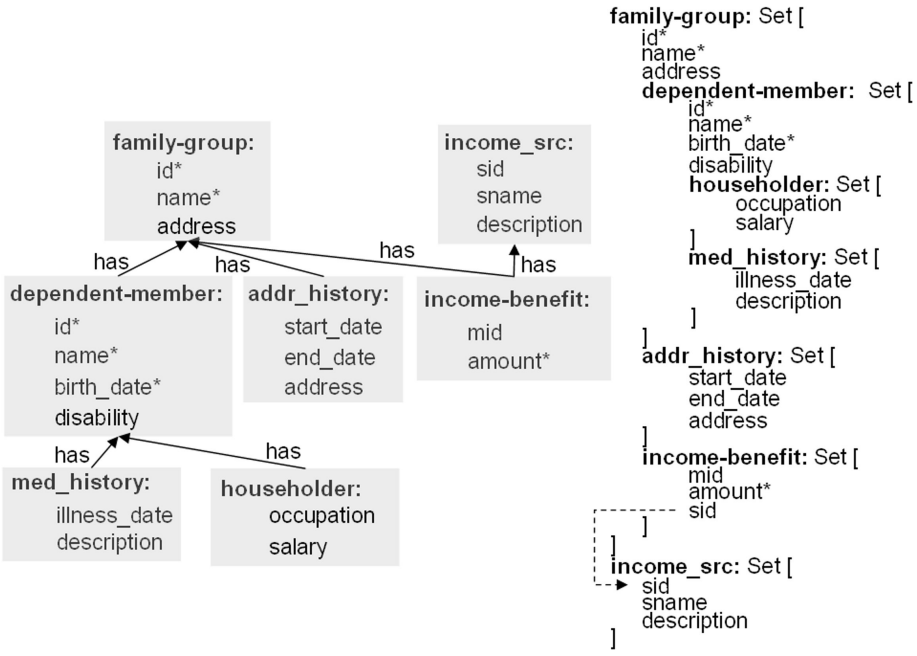

Figure 3.7: Combined concept graph and schema, for $\lambda = 0.37$

λ can be thought of as a tradeoff between precision and recall, decreasing λ increases precision at expense of recall and vice versa.

Intuitively, decreasing λ is appealing from database normalization perspectives. However, decreasing λ excessively is impractical and may result in ignoring edges connecting relatively similar concepts and hence, missing combining such concepts. So, the question is how can we decrease λ without affecting the edges used in the combination process.

3.5 Top-K Candidates

In Section 3.4 we described the basic approach for constructing an integrated schema using directed distances. The λ parameter was used to control the integration decisions, and the question of efficient tuning of λ was posed. In this section we address this question and present a revised approach that offers: 1) A systematic and efficient way of enumerating the top-k candidate integrated schemas, 2) using such candidates for automatic λ tuning, and 3) presenting the best integrated schema utilizing a stability analysis technique previously

proposed in [55] in the context of schema matching. The main steps of the approach are outlined in Figure 3.8, the gray boxes highlight the newly added steps. After constructing the concept graphs and calculating the directed distances between concepts across graphs, we consider the enumeration of candidate integrated schemas and formalize the notion of an assignment and its associated cost function. Then, a top-k enumeration algorithm follows. The top-k candidates are used to tune λ, they are also used in calculating the best assignment employing stability analysis. The best assignment is used to construct the integrated concept graph, which, in turn, will be used to construct the final integrated schema.

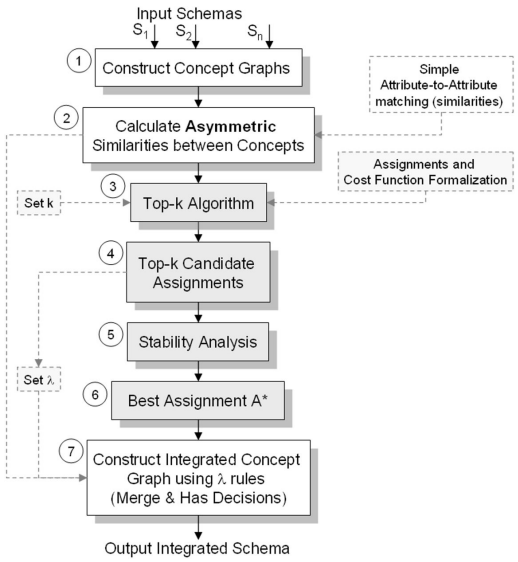

Figure 3.8: Revised workflow incorporating top-k candidates and stability analysis

3.5.1 Candidates Enumeration

Consider all possible integrated schemas that can be generated based on the exiting *distance_edges*. For each of these edges, we consider the alternative of either using that edge (to either merge the concepts or to add a has edge) or not using it at all. To consider all the alternatives for all the edges, we use bit assignments as follows. An $assignment\ vector\ \mathbb{A}$ is a fixed-sized, ordered vector of bits, where each bit corresponds to one $distance\_edge$.

When a specific bit is set to 1, this means that the corresponding $distance\_edge$ must be used. On the other hand, if it is set to 0, then the corresponding $distance\_edge$ will not be used. For each assignment, the input concept graphs are combined into an integrated concept graph by using the procedure described in Section 3.4 (by considering only the edges for which the bit is set to 1). Different assignments give rise to different ways of combining the concepts, thus yielding different candidate integrated schemas.

As an example, consider the concept graphs in Figure 3.2. Assume that the 8 *distance_edges* shown in the figure are our whole set of *distance_edges*. The $assignment\ vector$ can be represented as [X_7 X_6 X_5 X_4 X_3 X_2 X_1 X_0] (the variables are also shown above the corresponding *distance_edges* in the figure). There are 2^8 possible assignments; Figure 3.9 shows an example assignment [00001111] dictating the usage of the *distance_edges* corresponding to only X_0, X_1, X_2 and X_3.

In general, a naive enumeration of all assignments is highly ineffective due to their exponential number. The method developed in [33, 34] explores the space of assignments in an interactive way, where user constraints are used to direct the exploration of the space in an efficient way. Here, we adopt a different exploration strategy that is based on ranking the candidate schemas. Our first step is to aggregate the cost of an assignment (and of the resulting schema) from the individual distances of the edges in the assignment. Once we establish such cost, we develop a $top\ k$ algorithm that can generate the first k schemas in ranked order.

3.5.2 Cost of an Assignment

In order to compute an aggregated cost for an assignment, we need to consider first the costs associated with each individual decision of whether to use or not to use a $distance\_edge$. Assume $distance\_edge\ E$ connects concepts A and B. If E is used (i.e., its corresponding bit in the assignment is set to 1) then A and B will be combined (either by merging the concepts or by introducing a has relationship). Intuitively, the cost (or penalty) incurred

by taking the decision of using E should be proportional to the distance between the two concepts: $cost_E = \theta_1 \times \hat{D}(A, B)$. In other words, if two concepts are far apart (according to D) and we decide to combine those two concepts then this decision should be given a high cost (and vice versa, if the concepts are close to each other). Conversely, if E is not used (i.e., its corresponding bit in the assignment is set to 0). We take the cost (or penalty) incurred by this decision to be proportional to the similarity between the concepts: $cost_E = \theta_2 \times \hat{S}(A, B)$ The proportionality constants θ_1 and θ_2 can be used to give more or less weight for specific edges (in a user interaction mode). For the sake of simplicity, we shall assume that there is no such preference and $\theta_1 = \theta_2 = 1$.

Now we can define the aggregated cost for an assignment. Given an assignment $\mathbb{A}$ having a set K_1 of used edges and a set K_2 of unused edges, we define the cost of $\mathbb{A}$ to be:

$$cost(\mathbb{A}) = \frac{1}{n}\left(\sum_{E \in K_1} \hat{D}(A, B) + \sum_{E \in K_2} \hat{S}(A, B)\right) \tag{3.7}$$

where n is the total number of *distance_edges* in $\mathbb{A}$. Note that $cost(\mathbb{A})$ is always between 0 and 1, inclusive.

Figure 3.9 shows an example assignment $\mathbb{A}_i = [00001111]$ for the concept graphs of Figure 3.2. Applying Equation (3.7), $cost(\mathbb{A}_i) = \frac{1}{8}\ [\ (0.267 + 0.333 + 0.361 + 0.389) + (0.344 + 0.333 + 0.278 + 0.25)\ ] = 0.319$.

Not Used				Used				
X_7	X_6	X_5	X_4	X_3	X_2	X_1	X_0	
0	0	0	0	1	1	1	1	
0.25	0.278	0.333	0.344	0.611	0.639	0.667	0.733	$\hat{S}(A,B)$
0.75	0.722	0.667	0.656	0.389	0.361	0.333	0.267	$\hat{D}(A,B)$

Figure 3.9: Assignment cost calculation

We note that the cost model we adopt here is a very simple one, and it is quite conceivable that it can be refined to account for additional factors. For example, it could be further refined to cost differently the two ways of using an edge (i.e., merging vs. adding a has relationship) or it could be refined to account for how the schemas will be actually

used (by taking user constraints or query workloads into account). Nevertheless, the overall framework does not depend on the concrete choice of a cost function, as this cost function is a pluggable component. Given a cost function, the following $top\ k$ algorithm is able to find the best schemas (according to the cost function).

3.5.3 Top-K Algorithm

In general, assignment problems can be solved using, for example, the Munkres algorithm [99] or other related top-k algorithms [67, 101]. These papers show that assignment problems can be generally solved in polynomial time, in cases that satisfy a $(1:1)$ mapping cardinality constraint, which means in our terms, that one concept in one schema can only be combined with at most one concept in the other schema. This is a serious limitation since, in many practical cases, a concept in a schema can have correspondences (and can be combined) with multiple concepts in another schema. The study in [55] proposed formalizing schema matching problems as assignment problems. It gives an algorithm that, given two input schemas and a set of similarities between their elements, generates the top-k schema matchings, in cases where valid solutions satisfy a $(1:n)$ mapping cardinality constraint. This constraint means that a single element in the first schema can be mapped to multiple elements in the second schema, but a single element in the second schema can only be mapped to a single element in the first schema. This assumption is also a strong limitation and prevents us from using such algorithm.

We now start describing our algorithm for top-k enumeration. The initial step is to calculate the first, optimal assignment $\mathbb{A}_1$, which minimizes the cost. Let e_i be the correspondence represented by bit i in the assignment. Let $\hat{S}_i$ and $\hat{D}_i = 1-\hat{S}_i$ be the (undirected) similarity and distance associated with e_i. Based on the cost formulas in the previous subsection, if $\hat{S}_i \geq \hat{D}_i$, the cost of including e_i in an assignment is smaller than the cost of excluding it. Thus, the optimal assignment (of minimum cost) must necessarily have the bit i set to 1. Conversely, if $\hat{S}_i < \hat{D}_i$, the cost of including e_i in an assignment is larger than

X_7	X_6	X_5	X_4	X_3	X_2	X_1	X_0	
0.25	0.278	0.333	0.344	0.527	0.639	0.667	0.733	$\hat{S}$
0.75	0.722	0.667	0.656	0.473	0.361	0.333	0.267	$\hat{D}$
0	0	0	0	1	1	1	1	*k=1*
0.5	0.444	0.334	0.312	0.054	0.278	0.334	0.466	Δf

Sort Δf

X7	X0	X6	X5	X1	X4	X2	X3	Map	Δ
0.5	0.466	0.444	0.334	0.334	0.312	0.278	0.054	Δf_S	⇓
0	0	0	0	0	0	0	1	k=2	0.054
0	0	0	0	0	0	1	0	*k=3*	0.278
0	0	0	0	0	1	0	0	k=4	0.312
0	0	0	0	0	0	1	1	*k=5*	0.332
0	0	0	0	1	0	0	0	k=6	0.334

(a)

Assignment								*k*
X_7	X_6	X_5	X_4	X_3	X_2	X_1	X_0	
0	0	0	0	1	1	1	1	*1*
0	0	0	0	0	1	1	1	2
0	0	0	0	1	0	1	1	*3*
0	0	0	1	1	1	1	1	4
0	0	0	0	0	0	1	1	*5*
0	0	0	0	1	1	0	1	6

(b)

Figure 3.10: (a) Initial steps in the top-k enumeration process, (b) top 6 assignments

the cost of excluding it. Thus, the optimal assignment must necessarily have the bit i set to 0. As it can be seen, it is immediate to derive the optimal assignment $\mathbb{A}_1$. For our example, the assignment shown in Figure 3.9 is the optimal assignment.

The challenge is to efficiently derive the next $k-1$ best assignments. That is, given the jth assignment, we need to decide which bits to "flip" in the assignment in order to obtain the $(j+1)$th best assignment. Let us define $\Delta f_i = |\hat{S}_i - \hat{D}_i|$, to quantify the cost impact of flipping (i.e., complementing) the bit i from its current value in the optimal assignment $\mathbb{A}_1$. Let us define the vector $\Delta f = [\Delta f_{n-1}, ..., \Delta f_0]$. For our example, the top section of Figure 3.10(a) shows the optimal assignment (in the line with $k = 1$) and the vector Δ_f (in the next line). For each i, Δf_i represents the *increase in cost* with respect to $cost(\mathbb{A}_1)$ if the bit i (i.e., variable X_i) in $\mathbb{A}_1$ were to be flipped. We next sort the Δf vector in increasing order. Let us denote the sorted vector as Δf_s. Moreover, a vector Map keeps the association (after sorting) between elements of Δf_s and the variables X_i. See Figure 3.10(a) for an illustration of Map and Δf_s.

In order to generate the next best assignments, we explore, *incrementally*, modifications of the original assignment $\mathbb{A}_1$. The incremental modifications are based on the sorted vector Δf_s of cost increases. The goal, at each iteration, is to discover the next assignment that minimizes the increase in cost (as compared to the current assignment). To give an intuition, let us look at Figure 3.10(a) again. It can be seen that the 2nd best assignment (shown in the line for $k = 2$) can be obtained by just flipping the bit X_3, since this will give the least

cost increase (according to Δf_s). Next, to compute the 3rd best assignment, we need to change the variable with the next cost increase (i.e., X_2) and leave X_3 unflipped (relative to $\mathbb{A}_1$). To obtain the 4th best assignment, we have two choices. We could either flip variable X_4 (which is the variable with the next cost increase) and leave the rest as in $\mathbb{A}_1$, or flip both X_2 and X_3. In order to decide which choice gives the smaller cost increase, we need to actually calculate the costs for the two choices (i.e., 0.312 vs. $0.278 + 0.054$) and take the minimum.

As the example suggests, the key ingredients behind our algorithm are the following:

- To find the top k assignments, it is enough to consider combinations of flipping or not flipping the lower $k - 1$ variables in the Map vector.
- Furthermore, at any point in the algorithm, there are at most *two* new alternatives to be considered. To discover the next solution, it is enough to compare the new alternative(s) with the best candidate in a candidate set. The assignment with minimum cost increase becomes the next solution, while the rest will be re-evaluated in the next iteration.

In the algorithm, we make use of bit vectors of the form $F = [f_{n-1}, ..., f_0]$ to keep track of the bit flips as compared with the best assignment $\mathbb{A}_1$. Each vector F encodes an assignment as follows. If $f_i = 1$, then the variable in the ith position in the Map vector is flipped relative to its value in $\mathbb{A}_1$. Alternatively, if $f_i = 0$ then the same variable is left unchanged. The algorithm iteratively outputs the top-k assignments in increasing cost in the form of bit vectors $F = [f_{n-1}, ..., f_0]$. The actual assignments can be found immediately based on F, Map and $\mathbb{A}_1$.

Additional Vector Notation When describing a bit vector F, it is convenient to drop the most significant bits that are set to 0. We refer to such a representation as the *condensed* representation. We denote by $len(F)$ the size (or the length) of the condensed representation of F. As an example, the condensed representation of F = $[00000010]$ is $[10]$; moreover,

$len(F)$ = 2. (By convention, we take the condensed representation of a bit vector with all zeros to be $[0]$.) We also make use of a "reverse" operation that expands a bit vector. If F is a bit vector in condensed representation, and m is such that $len(F) \leq m$, we denote by $e_m(F)$ the *expansion* of F on m bits. Such expansion is obtained by adding 0's in front of the leading 1, so that the size of the resulting vector is m. Finally, we use $\|$ to denote the concatenation of a bit vector in front of another. For example, $[1] \parallel [01] = [101]$.

Algorithm 1 Enumerating the top k assignments

1: $soln \Leftarrow \{[0], [1]\}$
2: $cand \Leftarrow \{[10]\}$
3: **for** $3 \leq i \leq k$ **do**
4: $c = \min(cand)$
5: Output c as the i^{th} best solution
6: Remove c from $cand$
7: Add c at the end of $soln$
8: Write c, in condensed representation, as $[1] \parallel e_m(s)$. (Here, $len(c)$ is $m+1$, and s is the condensed representation of the "lower" m bits of c).
9: Let $next_m(s)$ be the first element s' in $soln$ that is after s and satisfies $len(s') \leq m$.
10: **if** $next_m(s)$ exists **then**
11: $candidate = [1] \parallel e_m(next_m(s))$
12: Insert $candidate$ in $cand$
13: **end if**
14: **if** $s = [0]$ **then**
15: $candidate = [10] \parallel e_m(s)$
16: Insert $candidate$ in $cand$
17: **end if**
18: **end for**

Algorithm 1 gives the exact steps for enumerating the top-k assignments. The two data structures used are: a list "$cand$" of candidates and a list "$soln$" of solutions generated so far. The list $cand$ is implemented as a priority queue, so that its operations (insertion, deletion, and taking the minimum) have all logarithmic complexity.

In lines 1 and 2, $soln$ is initialized with the top 2 assignments, in sorted order, and $cand$ is initialized to contain the 3^{rd} best assignment.[12] The **for** loop (lines 3-18) iteratively outputs the i^{th} best assignment starting from $i = 3$. In lines 4-7, the candidate c of minimum

[12]Recall that we manipulate F-vectors.

cost increase in $cand$ is output as the i^{th} solution, and then moved from $cand$ to the end of $soln$.

In lines 8 to 17, two new possible candidates are calculated based on the current best solution c and the solutions found so far. The bit vector c, assumed to have $m+1$ bits, is first decomposed into two parts: the highest significant 1, and the remaining lower m bits whose condensed representation is denoted by s. Next, we look for $next_m(s)$, which is the first vector s' in $soln$ that succeeds s (i.e., has higher cost) and satisfies $len(s') \leq m$. In particular, s' is the first higher cost solution (after s) that can still be written on m bits. Note that $next_m(s)$ may not necessarily be the immediate successor of s. Also, note that $next_m(s)$ may not exist.

In lines 10-13, if $next_m(s)$ is found, a new candidate is generated and added to $cand$. This new candidate is obtained by replacing s with $next_m(s)$ within the lower m bits of c. Additionally, in lines 14-17, if all the lower m bits of c are 0, a new candidate is generated by advancing the leading 1 by one position. Intuitively, these are the two possible ways of generating new candidate assignments, while minimally increasing the cost.

Example Let us follow the first three iterations of the algorithm for our running example (see Figure 3.10 again). The solution set $soln$ is initialized with $\{[0], [1]\}$, and the candidate set $cand$ is initialized with $\{[10]\}$.

- candidate $c = [10]$ is the only entry in $cand$ and, hence, it is the best. Thus, c becomes a solution (the 3rd), and it is moved from $cand$ to the end of $soln$. Two new candidates are placed into $cand$: $[11]$, obtained by replacing $[0]$ in c with $next_1([0]) = [1]$, and $[100]$, obtained by advancing the leading 1 in c. The candidate set is now $cand$ = $\{[11]_{0.332}, [100]_{0.312}\}$, where we write as a subscript the associated cost increase for each candidate (based on Δf_s).

- the best candidate c is now $[100]$, since $0.312 < 0.332$. Thus, c becomes the 4th solution, and it is moved from $cand$ to $soln$. As before, two new candidates are placed in $cand$: $[101]$, obtained by replacing $[00]$ in c with the expansion on 2 bits of

$next_2([0]) = [1]$, and $[1000]$, obtained by advancing the leading 1 in c. The set $cand$ is now $\{[101]_{0.366}, [1000]_{0.334}, [11]_{0.332}\}$.

- the best candidate c is now $[11]$. This entry is removed from $cand$ and becomes the 5th solution. Note that c, in this case, will not generate any new candidates. First, there is no $next_1([1])$ in $soln$, since all solutions following $[1]$ have len greater than 1. Moreover, c does not pass the test in line 14. The candidate set becomes $cand$ = $\{[101]_{0.366}, [1000]_{0.334}\}$ (and $[1000]$ will become the $6th$ solution in the next iteration).

Algorithm Analysis

Algorithm Correctness We can prove the correctness of the top-k algorithm by induction. For simplicity of the presentation, we shall assume that there are no ties among the assignments (i.e., no two assignments have the same cost). If ties occur, the same proof of correctness applies but with slight extensions (essentially, we must define a certain order among the assignments with the same cost, and then observe that the algorithm outputs ties in that order).

For $k = 1$, 2, and 3, the algorithm is correct by initialization. For $k > 3$, assume that the algorithm has generated the first $k - 1$ solutions $F_1, \cdots F_{k-1}$. We show that the solution with the k-th best cost, F_k, must be in the candidate set $cand$ at the beginning of the k-th iteration of the algorithm. Therefore, F_k will be generated as the k-th solution by the algorithm. Let us write F_k, in condensed representation, as $[1] \parallel e_m(s)$. We assume here that $len(F_k)$ is $m + 1$, and s is the condensed representation of the lower m bits of F_k. There are two cases to consider.

Case 1: $len(F_k) > max_{1 \le i \le k-1}(len(F_i))$. Since F_k is the first (i.e., lowest-cost) solution of its length, it must be that $s = [0]$. (We make use here of an obvious monotonicity property on assignments, and of the assumption of no ties.) Consider now the assignment $c = [1] \parallel e_{m-1}([0])$, which is the same as F_k except that the leading 1 occurs in a position

that is lower by one. Since $cost(c) < cost(F_k)$, it must be the case that c is one of the previous $k-1$ solutions. Hence, in an earlier iteration, the algorithm must have added F_k to $cand$ (lines 14-17).

Case 2: $len(F_k) \leq max_{1 \leq i \leq k-1}(len(F_i))$. Thus, there is an earlier solution F_l such that $len(F_k) \leq len(F_l)$. In turn, this implies that there is a solution F_i (with $i \leq l$) such that $len(F_i)$ is exactly the same as $len(F_k)$. This can be shown by observing that solutions do not "skip" any length number, that is, if there is a solution of length p then there is another (lower-cost) solution of length $p-1$. Essentially, the length can be reduced by 1 as follows: a vector of the form $[10\ldots]$ can be replaced by $[01\ldots]$, while a vector of the form $[11\ldots]$ can be replaced by $[01\ldots]$. In each case, the resulting vector has a lower cost.

Since $len(F_i) = m+1$, it must be that F_i is of the form $[1] \parallel e_m(s_i)$ where s_i is some bit vector in condensed representation such that $len(s_i) \leq m$. Note that both s and s_i must be among the top $k-1$ solutions, since their respective costs are smaller than F_k and F_i. Furthermore, since $cost(F_k) > cost(F_i)$, and F_k and F_i are identical on the leading bit, it must be that $cost(s) > cost(s_i)$. This implies that s_i appears in $soln$ before s. Let s' be the nearest solution in $soln$ that precedes s and has length smaller than m. We know that such s' exists, because s_i itself precedes s and has length smaller than m. Note that $next_m(s') = s$.

Let us now form the vector $F' = [1] \parallel e_m(s')$. Since $cost(s') < cost(s)$, we have that $cost(F') < cost(F_k)$. Thus, F' is one of the top $k-1$ solutions. This fact, combined with the fact that $next_m(s')$ exists, implies that in an earlier iteration, the algorithm must have added $[1] \parallel e_m(next_m(s'))$ to $cand$ (lines 10-13). But this last vector is precisely F_k. This concludes the proof.

Complexity Analysis First of all, the size of $cand$ is $O(k)$. This is because at each iteration, we take out one element and put at most two new candidates. So, the list increases by at most 1 at each iteration. Then, at each iteration, the complexity of finding the candidate with minimum cost (line 4) is $O(\log k)$ (given a priority queue implementation for $cand$).

Determining $next_m(s)$ requires $O(k)$ in the worst case, since it requires traversing the $soln$ list, and the size of $soln$ is at most k.

So, each iteration takes $O(k)$ and there are $O(k)$ iterations. Hence, the overall complexity is $O(k^2)$. Since the initial time to sort the Δf_s vector is $O(n \log n)$, we obtain an overall complexity of $O(n \log n + k^2)$.

Note that the cost of obtaining the next solution is only $O(k)$ at each step. This is important, since it is often the case that we do not need to generate all the top k solutions at once, but rather generate them lazily, one by one, as the user interaction demands it. In such case, the time to obtain the next solution is the important one.

3.5.4 Lambda Tuning Revisited

In Section 3.4.2, the question of efficient λ tuning was posed. Here, this question is revisited and an answer is proposed. We previously highlighted that tuning λ can be viewed as a tradeoff between precision and recall, and in fact, it is a function of how precise the attribute correspondences were calculated. Typically, such attribute correspondences may have been calculated using simple attribute-to-attribute schema matching techniques that examine types and properties of attributes. Also, this may involve calculating similarities based on available data instances for corresponding attributes. Accordingly, such process is inherently imprecise. For example, *social security number* can be represented as a *string* type in one schema but as an *int* type in another. Moreover, the number of available data instances for corresponding attributes can highly affect the overall precision, typically, a large number of available data instances will increase precision, and vice versa.

So, a good λ value for a specific integration problem may not be as good for another problem. From database normalization perspectives, it is appealing to decrease the value of λ, however, such decrease, may risk loosing some edges that are incorporated in the integration process. A straight forward approach for addressing such problem is to examine a *"representing sample"* for all *distance_edges* and tune λ accordingly. Having calculated

the top-k candidates, a possible choice for such sample is the set of all edges that were used in any of the top-k assignments.

Assume $\mathbb{E}$ represents the set of *distance_edges* used in any of the top-k assignments, denote the two directed distances associated with any of those edges as $\hat{d}_1$ and $\hat{d}_2$. A possible way for setting λ is: 1) Iteratively scan all the *distance_edges* belonging to $\mathbb{E}$, 2) record $\min(\hat{d}_1, \hat{d}_2)$ and add this value to a list L, and finally, 3) set λ to the maximum of the values in L.

The above procedure guarantees that the edges used in any of the top-k assignments are taken into consideration when setting λ. Note that, this is the minimum possible value for λ that can be obtained without ignoring any of those edges. Edges that don't belong to $\mathbb{E}$ are quite extreme edges that can be safely ignored in the process of tuning λ. Those extreme edges are connecting quite unrelated concepts that are unlikely to be included in the integration process.

Note that, the number of the top candidates k may affect λ tuning. Specifically, increasing k may increase λ if any of the additional candidates uses a new edge having $\min(\hat{d}_1, \hat{d}_2) > \lambda$. So, the problem of tuning λ was replaced by a new problem of tuning k. However, the stability analysis discussions in subsection 3.5.5 and the experiments analysis in Section 3.6 will provide an adequate argument supporting such problem transformation. From practical perspectives, it'll be shown that choosing a value for k is easier that choosing a value for λ.

3.5.5 Stability Analysis

In subsection 3.5.3, we efficiently calculated the top-k candidate assignments, which in turn, are translated into the top-k candidate integrated schemas. In an off-line setting, those candidates can be presented to the user for a manual choice of the correct assignment. This off-line setting is useful in some applications. However, for obvious reasons, this manual setting is highly inefficient in dynamic environments. Specifically, the introduction

of the semantic web, mashups and web-services integration applications underscore the importance of providing an automated integration solution. In this section we employ a stability analysis heuristic previously proposed in [55] and tested on real and synthetic data. The top-k assignments are analyzed and a best assignment A^* is proposed as a single solution, and hence, translated into a single integrated schema.

Assume $\mathbb{E}$ represents the set of *distance_edges* used in any of the top-k assignments. The stability analysis heuristic proposes counting how many times an $edges$ in $\mathbb{E}$ appears in the top-k assignments. An $edges$ that appears a sufficient number of times will be part of A^*, otherwise, it will be ignored. A threshold is used to quantify the sufficient number of times an $edges$ has to appear in the top-k assignments. More discussion about this will be presented in the experiments in Section 3.6.

3.6 Experiments

We evaluate our integration approach using a number of real world and synthetic scenarios. In Section 3.5 we showed that our top-k generation algorithm has low complexity, which makes it amenable for an efficient implementation. Complexity analysis showed that our schema enumeration approach significantly outperforms naive enumeration and previously presented approaches [33, 34, 55]. In this section we verify this on our implementation; we analyze the performance of generating the top-k schemas for a number of scenarios, and also evaluate the usability and effectiveness of the integration process based on a user study. Furthermore, we evaluate the tuning method for the integration parameter λ, by varying the number of top configurations and analyzing the effects on the final results. Our system is implemented using Java and the experiments were carried on a PC compatible machine, with Intel Core Duo processor (1.8GHz), running Windows XP and JRE 5.0.

3.6.1 Real Integration Scenarios

We test the performance and effectiveness of our method in a number of integration scenarios. The schemas used in these scenarios are: a relational and an XML schema representing gene expression experimental results (GENEX); two relational schemas from the Amalgam integration benchmark [94] for bibliographic data; two variations of the XML schema for the DBLP bibliography; the first schema in the Amalgam benchmark and one of the previous DBLP schemas; two variations of xml schemas, representing information about organizations, departments, projects and funding; two fragment schemas from the health insurance domain, representing families, householders, dependents and benefits; two variations of schemas, representing information about students, courses and enrollment; and a scenario for a 3-way integration where three XML schemas are used, each with a different nesting structure, representing information about departments, projects and employees, two XML schemas representing enterprise business objects (such as orders, and customers related to orders), one from SAP and the other one from the IBM WebSphere Business Integration (WBI) suite; a relational and an XML version of the Mondial database [97].

Figure 3.11 shows, for each scenario, the number of input concepts, the number of distance edges, as well as the number of input schemas. The top 10 assignments for each scenario are also shown together with the associated cost and the time needed to evaluate the assignment. Note that, we denote by $assignment\ edges$ the subset of *distance_edges* connecting concepts having at least one attribute correspondence. For presentation simplicity, The shown assignments use the $assignment\ edges$ and not the whole set of *distance_edges* (e.g., for the $Amalgam\text{-}DBLP$ scenario, we have 210 *distance_edges* but only 10 $assignment\ edges$). The bottom row shows the recommended λ value for each scenario based on the evaluated top-k assignments, this value will be used in the combination process to construct the integrated schema corresponding to an assignment. As previously discussed, the candidate schemas can be enumerated in a user interactive fash-

ion, or a single schema can be automatically generated based on the stable edges in the top-k assignments.

	Integration Scenario																							
	Dept-Org			Family-Group			Amalgam			Amalgam-DBLP			Genex			DBLP			Student			3-Way		
concepts	10			10			20			29			13			24			6			11		
distance edges	25			25			75			210			42			128			9			40		
Assign. edges	8			8			4			10			4			4			5			10		
Input schemas	2			2			2			2			2			2			2			3		
	Assign.	cost	time (ms)	Assign.	cost	time (ms)	Assign.	cost	time (ms)	Assign.	cost	time (ms)	Assign.	cost	time (ms)	Assign.	cost	time (ms)	Assign.	cost	time (ms)	Assign.	cost	time (ms)
k=1	01110100	0.349	0.2	01001011	0.319	0.2	1000	0.223	0.1	0000000111	0.355	0.2	1110	0.304	0.1	0001	0.228	0.1	10100	0.237	0.1	1100010001	0.348	0.2
k=2	01110101	0.369	0	01001010	0.347	0	1001	0.267	0	0000000110	0.355	0	1111	0.351	0	0000	0.228	0	10101	0.277	0	1100010000	0.348	0
k=3	01110110	0.369	0	01001001	0.354	0	1010	0.348	0	0000000101	0.355	0	1100	0.367	0	0011	0.386	0	10110	0.303	0	1100010011	0.356	0
k=4	01110000	0.383	0	01001111	0.358	0.1	1100	0.358	0	0000000100	0.355	0.1	1101	0.413	0	0010	0.386	0	10000	0.337	0	1100010010	0.356	0.1
k=5	01111100	0.388	0.1	01000011	0.361	0.1	1011	0.392	0	0000000011	0.355	0.1	1010	0.439	0	0101	0.405	0	10111	0.343	0	1100010101	0.359	0.1
k=6	01100100	0.390	0.1	01011011	0.361	0.1	1101	0.402	0	0000000010	0.355	0.1	0110	0.451	0	0100	0.405	0	11100	0.357	0	1100010100	0.359	0.1
k=7	01010100	0.390	0.1	01101011	0.375	0.1	0000	0.473	0	0000000001	0.355	0.1	1011	0.486	0	1001	0.436	0	10001	0.377	0	1100010111	0.367	0.1
k=8	00110100	0.390	0.1	00001011	0.378	0.1	1110	0.483	0	0000000000	0.355	0.1	0111	0.498	0	1000	0.436	0.1	11101	0.397	0	1100010110	0.367	0.1
k=9	01110111	0.390	0.1	01001000	0.382	0.1	0001	0.517	0.1	0000001111	0.372	0.1	1000	0.502	0.1	0111	0.564	0.1	10010	0.403	0	1100011001	0.370	0.1
k=10	01110001	0.404	0.1	11001011	0.382	0.1	1111	0.527	0.1	0000001110	0.372	0.1	0100	0.514	0.1	0110	0.564	0.1	11110	0.423	0.1	1100011000	0.370	0.2
λ value	0.500			0.667			0.500			0.500			0.571			0.771			0.667			0.5		

Figure 3.11: Results for different real world integration scenarios

To understand the shown assignments and what concepts are combined by each, Figure 3.12 shows, for each scenario, the set of $assignment\ edges$, and the corresponding directed distances for each, for example, the first row shows the ordered pair ($project$,$fund$) corresponding to the most significant bit in the assignment, and the ordered pair (0.667,0.750) as the directed distances. This means that the most significant bit in an assignment for the $Dept\text{-}Org$ scenario corresponds to the *distance_edge* connecting the concept $project$ in the first schema to the the concept $fund$ in the second schema, and, $\hat{d}(project,\ fund) = 0.667$, while $\hat{d}(fund,\ project) = 0.750$.

Figure 3.13 shows the characteristics of these scenarios, including the number of concepts, the number of correspondences between attributes, as well as the number of non-zero correspondences between concepts (i.e., correspondences between concepts that have non-zero similarity in at least one direction). In the fifth column of the table, we give the total number of integrated schemas (i.e., the size of the space of candidate schemas) in each scenario. The last portion of the table indicates the average time per generated schema when using the top-k schema integration method. This time is obtained by running the top-k algorithm for several values of k and then reporting the average time between two consecutive solutions (for each fixed k). Although the numbers of concepts and correspondences in these scenarios are not large, the schemas are not necessarily trivial. For example, the

	Assignment Map Distances
Dept-Org	(project,fund)(manager,emp)(dept,org)(emp,emp)(dept,address)(grant,fund)(manager,fund)(emp,fund) (0.667,0.750)(0.333,0.000)(0.333,0.000)(0.333,0.000)(0.167,0.656)(0.361,0.312)(0.583,0.500)(0.583,0.500)
Family-Group	(income_src,benefit)(dependent,member)(householder,benefit)(dependent,benefit)(family,group)(family,addr_fistory)(income,benefit)(householder,member) (0.667,0.750)(0.267,0.000)(0.722,0.500)(0.667,0.500)(0.333,0.000)(0.167,0.656)(0.361,0.312)(0.389,0.000)
Amalgam	(AUTHOR,AUTHOR)(MISC,UNPUBLISHED)(TECHREPORT,UNPUBLISHED)(ARTICLE,ARTICLE) (0.000,0.000)(0.769,0.500)(0.750,0.500)(0.417,0.588)
Amalgam-DBLP	(phdthesis,AUTHOR)(mastersthesis,AUTHOR)(book,BOOK)(mastersthesis,TECHREPORT)(inproceedings,INPROCEEDINGS)(article,ARTICLE)(phdthesis,BOOK)(author,AUTHOR)(author,AUTHOR)(author,AUTHOR) (0.900,0.500)(0.800,0.500)(0.571,0.750)(0.400,0.750)(0.375,0.583)(0.500,0.583)(0.500,0.583)(0.500,0.500)(0.500,0.500) (0.500,0.500)
Genex	(TL_FACTORVALUE,treat_factor)(EXPERIMENTFACTOR,exp_factor)(EXPERIMENTSET,exp_set)(TREATMENTLEVEL,treatment) (0.196,0.206)(0.214,0.229)(0.286,0.375)(0.571,0.594)
DBLP	(article,conf_jour)(article,year)(article,pub)(author,author) (0.917,0.500)(0.854,0.312)(0.817,0.771)(0.500,0.500)
Student	(underGrad,Student)(gradEnroll,eval)(enroll,course)(enroll,eval)(gradEnroll,Student) (0.000,0.000)(0.800,0.667)(0.083,0.250)(0.667,0.667)(0.600,0.333)
3-Way	(emp,emp)(dependent,dependent1)(proj,proj1)(dept,dept1)(proj,dept1)(project,proj1)(emp,emp1)(emp,emp1)(proj,project)(proj,dept) (0.167,0.167)(0.056,0.185)(0.500,0.667)(0.500,0.667)(0.500,0.667)(0.333,0.333)(0.167,0.611)(0.167,0.556)(0.125,0.542)(0.500,0.500)

Figure 3.12: Assignments Edges and the corresponding distances for the integration scenarios in fig. 3.11

Integration Scenario	Attribute Corresp.	Source Concepts	Concept Corresp.	Total Integr. Schemas	Time/schema (ms) (top-64)	(top-128)	(top-256)
Genex	31	13	6	64	0.50	n/a	n/a
WBI-SAP	46	22	7	128	1.70	2.44	n/a
Mondial	74	49	18	>3000	1.43	1.72	2.13
Amalgam	16	24	12	>3000	0.95	1.12	1.43
Amalgam-DBLP	26	29	11	2048	0.85	0.98	1.10
Dept-Proj-Emp	16	10	8	192	0.22	0.25	n/a

Figure 3.13: Evaluation on real schema integration scenarios.

SAP schema is split over 15 files with concepts having tens (20-50) of attributes and multiple levels of nesting. Moreover, many of the concepts are referred to from other concepts. One reason for why there is only a small number of concept correspondences (seven in the WBI-SAP integration scenario) is that not every concept in one schema has a match in the other schema. Nevertheless, the size of the space of candidate schemas in these scenarios is sufficiently large to illustrate the need for an effective tool to explore all the integration choices. For larger schemas, the effectiveness of our top-k enumeration method would become even more apparent. As it can be seen, the average time to generate the next schema is consistently low, even when k is large (e.g., 128). Note that in practice, in an interactive

system, we will seldom need to generate such a large number of schemas (see also the next subsection).

We also observed from the experiment that the top schemas generated by the algorithm make the "correct" decisions for a large fraction of the bits (or, variables) in the assignment vector. In particular, the highly similar concepts (with many attributes in common) are combined, while the highly dissimilar concepts are not combined. In a sense, these are the "easy" decisions that an expert may often agree with. This leaves out the "difficult" cases, involving groups of "ambiguous" concepts for which any combination may be possible.

To illustrate, consider the Mondial scenario. There are 18 correspondences between concepts. Most of them (14, precisely) involve concepts that are semantically the same (e.g., *river*, *sea*, *lake*, *city*, which occur in both input schemas). However the concept $country$ in one of the schemas can be combined with any subset of COUNTRY, ECONOMY, and POPULATION in the other schema. This is reflected in the respective similarity numbers which are very close to 0.5. The reason for this "ambiguity" is that $country$ in the first schema is an unnormalized relation that includes attributes specific to economy and population, which are stored separately, in different relations, in the second schema. The schemas at the top of our ranking will enumerate all possible choices of combining these concepts (from the most normalized to the most unnormalized), while making the obvious choice for the rest of the concepts. In contrast, the schemas that are at the bottom of the ranking will not combine some of the concepts that are obviously equivalent.

In the next subsection, we shall make a more precise qualitative argument, via a user study.

Our top-k enumeration algorithm performed well as expected from its complexity analysis, taking ~ 0.1 milliseconds on average to generate the next candidate assignment. The initial steps and finding the first assignment are relatively more expensive, taking on average ~ 0.2 milliseconds, since they involve sorting the whole set of *distance_edges*. Finding the immediate successive candidates is extremely fast (taking negligible time) since

we only calculated the top-10 assignments. The previously discussed complexity analysis confirms that our algorithm is practically efficient even if calculating a large number of k candidates is required.

3.6.2 User Experiment

In general, the best assignments generated by the top-k algorithm may not always yield the schema that is "best desired" by a particular expert, although they may yield a good approximation. Thus, a human expert is still required for the schema integration process. Our goal is to minimize the human effort rather than eliminate it. In this subsection, we shall evaluate the effectiveness of our top-k algorithm as part of an *interactive tool* that helps a human expert reach the "best desired" schema.

The tool works as follows. It first generates the (unconstrained) top-k schemas according to the initial input (schemas and correspondences). The user then inspects a few of the top schemas (the first, the second, etc.) and adds, if needed, one or more *constraints* in the sense of [34]. These constraints are a mechanism through which a human expert can "fix", or override, the undesired choices made by the top schemas. Next, the system reacts by generating the (new) top-k schemas that satisfy the constraints. The process continues, with the user possibly adding more constraints, and so on, until the "best desired" schema is obtained. There are two types of user constraints that we allow: "Always use correspondence X", and "Never use correspondence X".[13] The first constraint requires that the pair of concepts connected by correspondence X must always be combined. It is equivalent to fixing the bit for X (in the assignment vector) to be always 1. Conversely, the second constraint requires the correspondence X to be ignored. Thus, it is equivalent to fixing the bit for X (in the assignment vector) to be always 0. Note that, with each such constraint, the space of the candidate schemas is reduced by half. The regeneration of the top-k schemas is done by re-invoking the top-k algorithm, after taking out the fixed variables from the

[13] These constraints are called, respectively, $Apply(X)$ and $\neg Apply(X)$, in [34].

assignment vector. We note that, in the tool, we generate the top-k schemas only one at a time, as demanded by the user. (Thus, the time per schema is more important than the time to generate all top-k schemas.)

To evaluate the effectiveness of the resulting tool, we conducted a *user study* to evaluate the number of user interaction steps needed to reach the "best desired" schema. We had four users, which are researchers in our department and are database experts. Nevertheless, they required a few hours to go through the three scenarios we selected for this experiment, since they needed to understand the semantics of the schemas (and the domain). The results of our user study are summarized in Figure 3.14.

	Genex		Mondial		Dept-Proj-Emp	
	(constr.)	(schemas)	(constr.)	(schemas)	(constr.)	(schemas)
User A	1	2	1	2	2	3
User B	1	2	1	2	1	2
User C	1	2	3	4	2	3
User D	0	3	3	4	0	5

Figure 3.14: User study in three of the scenarios.

We measure two types of interactive steps in the experiment. First, we count the total number c of constraints that a user has to add, to instruct the tool. Second, we count the total number s of *different* schemas that the user explores, before deciding the final integrated schema. These numbers are not an exact measure of the human effort involved, since they do not include the time to understand the source schemas, or the time to evaluate the integrated schemas. Still, these parameters give an indication of the complexity of the interactive process. As it can be seen, the number of interaction steps is consistently low in all scenarios for all users.

For Genex, the users were consistent and picked the same "best desired" schema. A total of 4 out the 6 correspondences connected semantically equivalent concepts (with many common attributes). The top schemas generated by the tool chose, correctly, to merge these concepts. The users were left to decide how to combine the concepts of $array$ and

$measurement$ in one schema, with the concept of $array\_measurement$ (which included features of both array and measurement) from the other schema. All users decided that a good design is to leave $array$ as a separate concept, and to merge $array\_measurement$ with $measurement$. However, they accomplished this in different ways. User D let the tool enumerate the first few schemas and stopped when she realized that the 3rd schema was the desired one. Users A, B and C looked at the first schema, which merged the above three concepts into one, and concluded right there that they did not want $array$ to be merged with $array\_measurement$. Hence, they added a constraint to prevent such merging. The top schema (after regeneration) was then the desired schema.

The Mondial scenario had similar characteristics with Genex. In particular, the "hard" choices involved the $country$ concept (as discussed earlier). Here, the final schema was not the same for all the users. Two of them (C and D) chose to merge all four concepts that had country features (i.e., $country$, COUNTRY, ECONOMY, POPULATION), by explicitly adding three constraints. The other two (A and B) achieved a more normalized schema, by merging $country$ with COUNTRY, but leaving ECONOMY and POPULATION as separate concepts (with has edges to the integrated concept for country). Only one constraint was needed for this.

Finally, the scenario with more "ambiguous" choices was Dept-Proj-Emp, where most of the concepts in each schema could match several of the concepts in the other schema. For example, $employee$ in the first schema could match both $manager$ and emp in the second schema, while $fund$ in the first schema could match both $grant$ and $project$ in the second schema. Users A and D reached the same final schema, although in different ways. Users B and C decided on different schemas. Thus, different users can often have different integrated schemas (and different criteria of goodness) in mind. This confirms, one more time, that an automatic system for schema integration is not realistic. Even so, the tool was helpful in identifying the choices that all the users agreed with (e.g., merge $employee$ with emp, and merge $fund$ with $grant$).

Notably, the schema chosen by User C in this final scenario showed a case where the unconstrained top-k algorithm could not have generated the "best desired" schema (even for a large k). The reason for that is that User C decided that $fund$ should be merged with $grant$ and, additionally, with $project$. The reasoning was that usually there is a one-to-one relationship between a project and a grant; hence, the two concepts could be combined into one. On the other hand, the unconstrained top-k algorithm gives little weight to the choice of merging $fund$ with $project$ since they only have one attribute in common (project name).

Overall, the user study validates the idea that the top-k algorithm identifies many of the correct choices in practice. Thus, the user can focus her attention on the "difficult" cases. This is reflected in the number of user constraints needed to guide the system. This number is relatively small in our experiment.

3.6.3 Synthetic Scenarios for Analyzing Stability

We used a set of synthetic integration scenarios to study the frequency of used edges in the top-k assignments, and specifically, to inspect how such frequency changes with varying k. Intuitively, edges frequently used in the top-k assignments are more likely to be part of the best assignment, while rarely used edges (or totally unused) are less likely to be part of the best assignment. So, As previously outlined, we embraced the approach of generating the top-k candidates and analyzing the frequency of used edges, and hence, automatically calculating the best integrated schema.

If the frequency of used edges in the top-k assignments will be the main factor in deciding the best assignment, then it is extremely important to study how such frequency changes when we vary k. Typically, we need such frequency to quickly stabilize, in other words, we seek a characteristic where the increase in k doesn't significantly affect the calculated frequencies. Such characteristic will provide a justification for the decision of

relying just on a few number of top candidates to evaluate the frequencies. In the following, we shall show that our cost formalization offers such appealing characteristic.

We generated 100 synthetic integration scenarios, for each, we randomly varied the number of input schemas between 2 and 10, then, for each schema, we randomly generated a number of concepts between 3 and 20. And, finally we randomly generated the distances associated with all *distance_edges*. We used a uniform probability distribution function in all random number generations. For each scenario, we calculated the top-k assignments for $k = 1$, 10, 20 ... 100. For each top-k, we calculated the frequency of each used *distance_edge* and then calculated the absolute difference between such frequency and the frequency of the same edge in the preceding top-k group, and then calculated the average for all used *distance_edges*.

Figure 3.15 depicts such analysis, the value corresponding to $k = 10$ is the average difference in frequencies between the top-10 candidates and the top-1 candidate, while the value corresponding to $k = 20$ is the average difference in frequencies between the top-20 candidates and the top-10 candidates, and so on. It is clear from Figure 3.15 that as k increases the differences in frequencies become less significant. This supports our cost function formalization and the choice of relying on the enumeration of few top-k candidates.

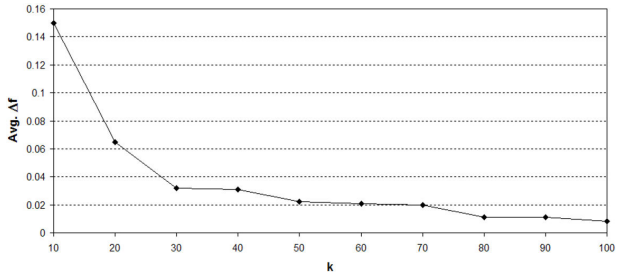

Figure 3.15: Average change in the frequency of the used edges in the top k schemas for different k values

3.7 Summary of Contributions

There are a number of features that distinguish our approach from existing work on schema integration, model merging and ontology merging. In the context of ontology merging, most literature has been primarily focused on the problem of ontology alignment, which is deriving relationships across concepts in different ontologies (see ILIADS [140] as an example). In contrast, the focus of our method is on the ranked exploration of the alternatives for the structural, non-redundant unification of the concepts.

We briefly visited the method of Pottinger and Bernstein [111] in Section 1; this method subsumes much of the earlier work on schema integration [22, 26, 128] and also includes merging-specific features that are present in PROMPT [102] and other ontology merging systems, such as FCA-Merge [131]. In the method described in [111], the user must provide in advance a "template" of the integrated schema. In contrast, the input to our method is just a set of atomic correspondences which can be discovered by an automatic schema matching tool. From such input, our system is then able to generate the most "likely" candidate integrated schemas. As in [111] and following [34], our method operates at a logical level, where the schemas are described in terms of concepts and their relationships.

The work of [111] is extended in [112] by considering the schema integration problem in the context of source schemas related by GLAV mappings, as opposed to just correspondences between schema attributes. One possible direction is to investigate whether our ranking and enumeration mechanism can extend to such context.

Our method builds upon the framework introduced in [34], which reduces the schema integration problem to an enumeration of all possible ways of merging two graphs of concepts. This enumeration defines in a very precise, mathematical sense, the space of candidate schemas. While the system in [34] relies exclusively on user interaction to navigate and explore the space of candidate schemas, the method we propose in this thesis is more automatic and relies on top-k enumeration to give higher priority to the more "likely" schemas. In particular, we keep and exploit all the weights generated by the matching al-

gorithms, which enables us to cost the integrated schemas. Furthermore, the use of weights also enable us to define more refined relationships on the integrated schema (i.e., incorporate both *merge* and *has* decisions).

Our formalization of the schema integration problem as a top-k enumeration is different from the general assignment problem [99], and existing techniques (see [99, 67, 101, 55]) do not immediately extend to our context. In particular, as discussed in Section 3.5.3, they limit the ways in which concepts in one schema can be combined to concepts in another schema.

Finally, we note that schema matching techniques have been extensively studied [118, 93, 25]. Our schema integration method is complementary to schema matching, since it uses the outcome of schema matching. Furthermore, the emphasis here is on using directed similarities rather than the more common undirected similarities.

Chapter 4

Bioinformatics Applications

In this section we focus on some important bioinformatics applications that required combining data from multiple sources. We describe our experience in collecting, preparing and integrating the required data. After going through the detailed schema integration steps, we investigate how these combined data can be used in studying a number of useful bioinformatics applications. The problems involved accessing and utilizing data from various data sources, for example, Genome DNA sequences from NCBI [21], Genes information (location, density, etc.) and DNaseI hyper-sensitive sites from the UCSC genome browser repository [79], micro-array and tissue specificity data from GNF SymAtlas [132], etc. We start by giving a background about the addressed applications, then we discuss the details of data collection, preparation and integration, followed by describing various methods used in these applications, and we demonstrate and discuss the results. Various applications are covered in this chapter, including a genome-wide nucleosome exclusion landscape, studies of gene density, tissue specificity, gene expression levels and DNaseI Hypersensitive sites in the human genome. Additionally we validate the *in silico* computational results we calculated with real laboratory results.

4.1 Background

Nucleosomes are DNA-protein complexes that form the building blocks of eukaryotic chromatin. They are involved in genome condensation, and play a major role in the regulation of gene expression [87]. Each nucleosome is made up of eight histone proteins that together form a structural unit able to accommodate 147 base pairs of DNA wound around it. The DNA sequence has to have the flexibility and curvature that allows it to circle around a nucleosome [145]. Empirical and theoretical studies have both shown that there are certain DNA sequence patterns that are too rigid to form such loops [139]. These patterns include GC-rich motifs as well as poly-A and poly-T tracts, these were compiled into NXSensor, a web tool that predicts which DNA sequences would not be conducive to nucleosome binding; these motifs are called *nucleosome exclusion sequences* [89].

Transcriptional regulation in eukaryotes is a complex process, as exemplified by recent publications (for example, [30, 38]). The formation and positioning of nucleosomes are crucial steps in gene regulation, in that they influence access to DNA by the transcriptional machinery. Experimental work on nucleosome positioning in yeast [149, 121, 85, 15, 16] and fly [113] has yielded significant results, and technological progress is such that we are quickly learning more about nucleosome positioning in the human genome [104, 59].

Experimental work that verifies where a nucleosome is positioned is dependent upon when the cells were sampled, and on which tissue or cell line the analysis was carried out. In addition, it is known that nucleosomes slide to allow certain regulatory mechanisms to take place [61], and it has been shown in yeast that nucleosomes are only occasionally positioned by intrinsic sequence signals [108]. We therefore chose nucleosome exclusion sequences as our predictive method, rather than nucleosome positioning sequences. We can with a certain level of certainty predict where nucleosomes would not bind, and it is therefore inferred that they can, and probably do, bind elsewhere.

Reported studies in [57, 89] observed certain trends in the nucleosome exclusion patterns of promoter regions. Both studies showed that there is a peak of nucleosome exclusion

sequences just upstream of the transcriptional start site of genes. This pattern has subsequently been found to be true also in yeast [85]. The studies in [57, 89] found that widely expressed genes, sometimes referred to as housekeeping genes, had a higher nucleosome exclusion potential than did tissue specific genes. This implies that the promoter regions of widely expressed genes were less likely to have nucleosomes in them than were the promoter regions of genes that had a narrow tissue distribution. This may allow easy access of the transcriptional machinery to the DNA of ubiquitously expressed genes. However, these studies had taken relatively small numbers of carefully selected human genes: 100 of each category in the case of [89], and 500 each in [57], and they both relied on manual selection and categorization of genes. The question remained whether there is a genome-wide trend of a gradient of nucleosome positioning potentials, and what implications this may have for the specificity of gene expression. These were the initial questions that we set out to answer in this study.

One objective of the present study, therefore, was to carry out a whole genome annotation of nucleosome exclusion regions (NXRs) in the human genome, and to correlate the results with tissue specificity, gene expression levels, and DNaseI hypersensitive sites. We calculated nucleosome exclusion scores (NXScores) across the whole genome, and observed NXScore trends in promoter regions. We classified tissue specific and widely expressed genes according to a new method proposed here based on Grubbs' outliers test, and validated the results using a previously described method based on Shannon's entropy [120]. From a computational perspective, patterns such as NXRs and NXSs are fuzzy, non-exact, and overlapping, which poses a challenge for the analysis of all 3.4 billion base pairs of the whole human genome. We therefore developed a pilot grid architecture that can carry out such computationally intensive tasks. In this study we report our results in the context of the regulation of gene expression.

4.2 Data Collection

The applications addressed in this section, not different from typical bioinformatics applications, required collecting, processing and combining data from various data sources. In the following, we'll describe these data sources in some detail and show how they were processed and combined to offer a convenient and efficient way of conducting the study. Schema integration offered a way of defining a unified representation for these data sources, and hence facilitating data processing and manipulation tasks.

First, the study required processing the whole human genome DNA sequence. *FASTA*[14] is a standard and popular format for representing and storing DNA sequences. The DNA sequences in FASTA format for chromosomes 1 to 22, in addition to chromosomes X, Y and M were downloaded from the NCBI data repository [21]. These sequences were processed to locate nucleosome exclusion regions and to calculate the associated nucleosome exclusion scores (the details for these processing steps is addressed in Section 4.4). The resultant nucleosome exclusion regions were stored using the schema described in Table 4.1, and the associated scores used the schema described in Table 4.2. Table 4.1 shows the attributes of the Nucleosome Exclusion Regions schema, together with their SQL types and descriptions for each. Each region has a unique id generated by the processing module to uniquely identify such region, the $chrom$ attribute shows which chromosome this region belongs to. $chromStart$ and $chromEnd$ identify the start and end positions for the region. The location is zero based (i.e. the first base pair in each chromosome is numbered 0). Since there are a number of different nucleosome exclusion patterns, the $type$ attribute is typically an enumeration of possible pattern types, specifically, $type - A$ identifies the $[(G/C)_3N_2]_{\geq 3}$ patterns, $type - B$ identifies the poly-A patterns and type C identifies the poly-T patterns (details on the choice and nature of these patterns are addressed later in Section 4.4). Lastly, the sequence attribute carries the actual sequence localized. Table 4.2 shows the schema for Nucleosome Exclusion Scores, where the $chrom$ attribute identifies

[14] http://www.ncbi.nlm.nih.gov/blast/fasta.shtml

Table 4.1: Nucleosome exclusion regions

Attribute	*SQL Type*	*Description*
id	varchar(20)	nucleosome exclusion region id
chrom	varchar(255)	chromosome
chromStart	int(10) unsigned	Start position in chromosome
chromEnd	int(10) unsigned	End position in chromosome
type	varchar(255)	pattern type (e.g. poly-A)
sequence	longblob	dna sequence for this region

Table 4.2: Nucleosome exclusion scores

Attribute	*SQL Type*	*Description*
chrom	varchar(255)	chromosome
chromStart	int(10) unsigned	start position in chromosome
chromEnd	int(10) unsigned	end position in chromosome
score	double	exclusion score

the chromosome, $chromStart$ and $chromEnd$ identifies a constant-score region within this chromosome (can be of variable length), and the $score$ shows the exclusion scores. For data size considerations, this table only contains records for regions having $score > 0$, a score of 0 is assumed for missing regions.

Calculating the nucleosome exclusion regions and the associated scores were two of the initial steps in this study. Next, it was required to combine these data with data from various other sources. The correlation with gene density required information about genes, their location and transcription information, etc. Table 4.3 shows the schema for the table having these information, the name attribute carries the gene id, the chrom attribute identifies the chromosome where this gene is located, the strand identifies if the gene is located on the +ve or -ve strand of the chromosome, txStart and txEnd specifies the transcription start and end positions, cdsStart and cdsEnd are the translation start and end positions, translation is the process of synthesizing the gene product (i.e. the protein). A description of the rest of the attributes in included in Table 4.3. The source for these data was the UCSC Genome Browser [79].

The correlations with tissue specificity and genes expression levels required incorporating micro-array and gene expression information and additionally to study how such expression levels vary from one tissue to the other. This required incorporating the GNF-SymAtlas datasets, which contains 44,775 expression profiles across 79 human tissues and

Table 4.3: Genes based on RefSeq, GenBank, and UniProt

Attribute	*SQL Type*	*Description*
name	varchar(255)	name of gene
chrom	varchar(255)	reference sequence chromosome or scaffold
strand	char(1)	+ or - for strand
txStart	int(10) unsigned	transcription start position
txEnd	int(10) unsigned	transcription end position
cdsStart	int(10) unsigned	coding region start
cdsEnd	int(10) unsigned	coding region end
exonCount	int(10) unsigned	number of exons
exonStarts	longblob	exon start positions
exonEnds	longblob	exon end positions
proteinID	varchar(40)	UniProt display ID for Known Genes, UniProt accession or RefSeq protein ID for UCSC Genes
alignID	varchar(255)	unique identifier for each (known gene, alignment position) pair

Table 4.4: Known to GNF Atlas2 (Genes' ids and the corresponding micro-array probe id

Attribute	*SQL Type*	*Description*
name	varchar(255)	primary id (Gene id)
value	varchar(255)	associated id (Probe id)

cell types and is itemized by oligonucleotide probes [133]. Table 4.4 shows the schema for the data source relating the genes and the corresponding micro-array probes. First, all non-specific and partially-specific micro-array probe sets were removed from our datasets, leaving only the specific target data, with each probe corresponding typically to only one gene. This probe set was then joined with the Known Gene database of the UCSC Genome Browser [79] (discussed above), from which information on the chromosomal location of the gene, and its transcriptional start and end positions was extracted. Any further redundancies were filtered out at this step, resulting in 19055 genes.

Table 4.5 shows the schema for the table having the information about expression scores (across tissues) corresponding to each micro-array probe. It shows the probe name, and the number of scores reported and a comma delimited sequence of scores. To link each indexed score in this sequence to the corresponding tissue name, the tissue indexes table is used (the schema is shown in Table 4.6). The correlation with the DNase1 hyper sensitive sites required incorporating an additional data source from the UCSC Genome browser repository. The schema is depicted in Table 4.7, this data source gives the values for the predicted hydroxyl radical cleavage intensity on naked DNA for each nucleotide. The promoter region sequence (-1500 to +500) for each of the genes involved in the study were downloaded and fed for analysis through processing and pattern matching modules.

Table 4.5: Tissues' expression scores per probe

Attribute	*SQL Type*	*Description*
name	varchar(255)	probe name
expCount	int(10) unsigned	scores count
expScores	longblob	reported scores (comma separated)

Table 4.6: Tissues' indexes and names

Attribute	*SQL Type*	*Description*
id	int(10) unsigned	tissue id (index)
name	varchar(255)	tissue name
description	longblob	
url	longblob	
ref	longblob	
credit	longblob	
numExtras	int(10) unsigned	
extras	longblob	

Table 4.7: Predicted hydroxyl radical cleavage intensity on naked DNA for each nucleotide in the ENCODE regions

Attribute	*SQL Type*	*Description*
bin	smallint(5) unsigned	indexing field to speed chromosome range queries
chrom	varchar(255)	reference sequence chromosome or scaffold
chromStart	int(10) unsigned	Start position in chromosome
chromEnd	int(10) unsigned	End position in chromosome
name	varchar(255)	name of item
span	int(10) unsigned	each value spans this many bases
count	int(10) unsigned	number of values in this block
offset	int(10) unsigned	offset in File to fetch data
file	varchar(255)	path name to data file, one byte per value
lowerLimit	double	lowest data value in this block
dataRange	double	lowerLimit + dataRange = upperLimit
validCount	int(10) unsigned	number of valid data values in this block
sumData	double	sum of the data points, for average and stddev calc
sumSquares	double	sum of data points squared, for stddev calc

Different data processing modules were wrapped as web services. The development was done using Java, and some modules utilized BioJava APIs [2].

4.3 Schema Integration

As discussed earlier in Chapter 3, the integration process starts by matching individual schema components, followed by integration decisions on how to structurally compose the integrated schema. Figure 4.1 shows the input (source) schemas (detailed description of these schemas can be found in Section 4.2). The source schema $S1$ represents the calculated data for nucleosome exclusion regions and scores, it shows three tables, the $Chromosome$, the $NXRegins$ and the $NXScores$ tables. The source schema $S2$

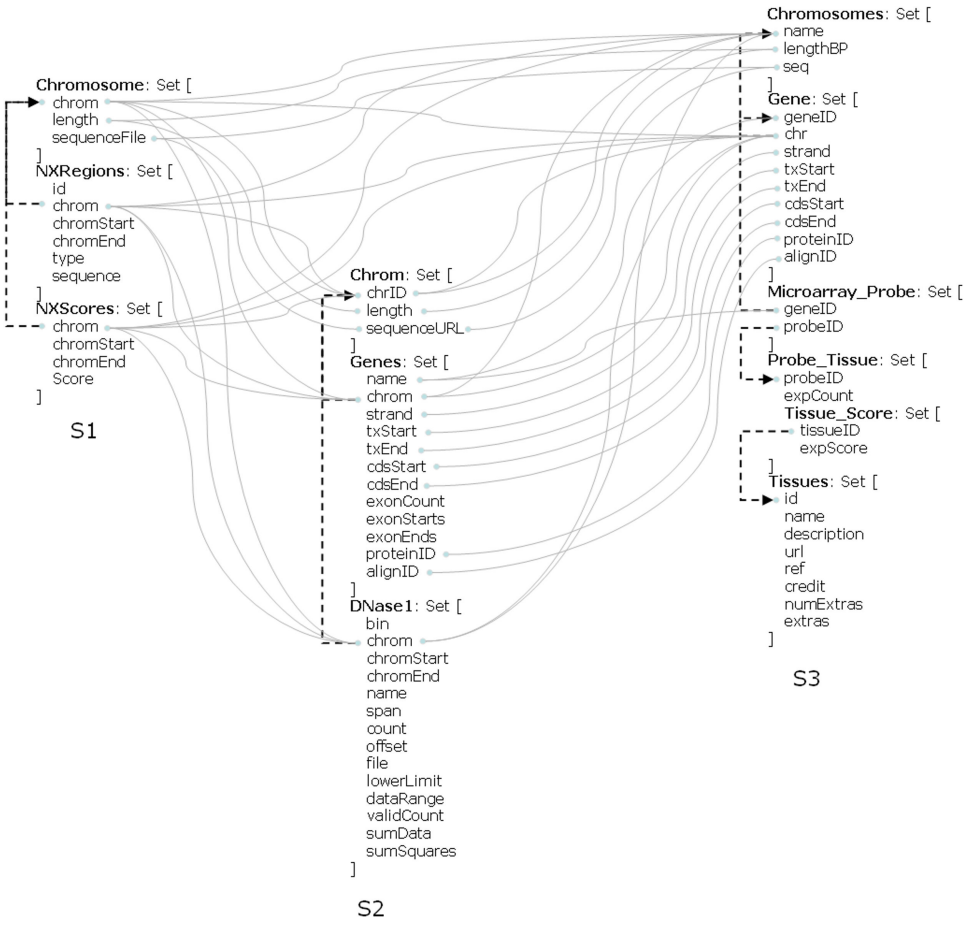

Figure 4.1: Input (source) schemas for the prepared data (S1), UCSC Genome Browser (S2) and GNF Atlas (S3).

represents the $Genes$ and $DNase1$ tables from UCSC Genome Browser, in addition to an alternative representation for chromosomes data (table $Chrom$). The GNF-SymAtlas data represented in source schema $S3$ shows data of tissues, genes and their expression scores (the $Microarray\_Probe$, $Probe\_Tissues$ and $Tissues$ tables), in addition to a source-specific alternative representatives for the genes and chromosomes information (tables $Chromosomes$ and $Gene$). The dotted directed lines represent foreign key/primary key relations, while the solid lines connect matching attributes across schemas (according to the schema matching process). The matching process relies on both the metadata (column names and types), in addition to similarities between actual data instances that these columns contain.

Having this representation for the input source schemas, the next step is to construct their corresponding concept graphs. Figure 4.2 shows the concepts graphs corresponding to the input schemas shown in Figure 4.1. The concept graphs corresponding to $S1$ and $S2$ has three concepts each, while the concept graph for $S3$ has six concepts. Refer to Section 3.2 for the details on how these concepts graph are constructed from their corresponding schemas. Concept graphs are used as an abstract model for representing different types of schemas and they serve as an intermediate representation that facilitate the schema integration process. Using our proposed schema integration approach, the integration process is a matter of evaluating possible combination decisions, and ranking these alternatives. So to qualify these decisions, the next step is to calculate the distances/similarities between the concepts across the input schemas. The procedure described in Section 3.3 is used to calculate the directed similarities shown in Figure 4.3. As previously mentioned, we only include similarities between concepts that has one or more matching attributes. The figure uses the graph and concept labels from Figure 4.2 as identifiers, for example, $S1.C1$ represents the concept labeled $C1$ in the $S1$ concept graph. In Figure 4.3, the tables (a), (b) and (c) show the edges connecting concepts across the $S1 \longleftrightarrow S2$, $S1 \longleftrightarrow S3$ and $S2 \longleftrightarrow S3$ schemas, respectively. Note that, for $S3$ only $C1$ and $C2$ are included, since the rest of the concepts have no matching attributes to any other concepts in $S1$ and $S2$. For each edge the table shows the two directed similarities (shown as an ordered pair between brackets), in addition to the undirected similarity (the min of both as described in Section 3.3). Here we have 21 distance edges, labeled X_0 to X_{20} and numbered based on descending order of their undirected similarities.

Having the edges and their corresponding directed and undirected similarities, we can use our Top-K algorithm (described in Section 3.5) to find the top combination candidates. The steps are depicted in Figure 4.4. The process starts with an initial step of calculating the first assignment $\mathbb{A}_1$, which is the minimum cost assignment, by using the edges where $\hat{S}_i \geq \hat{D}_i$ (to minimize the overall cost of the assignment). The next step is to calculate

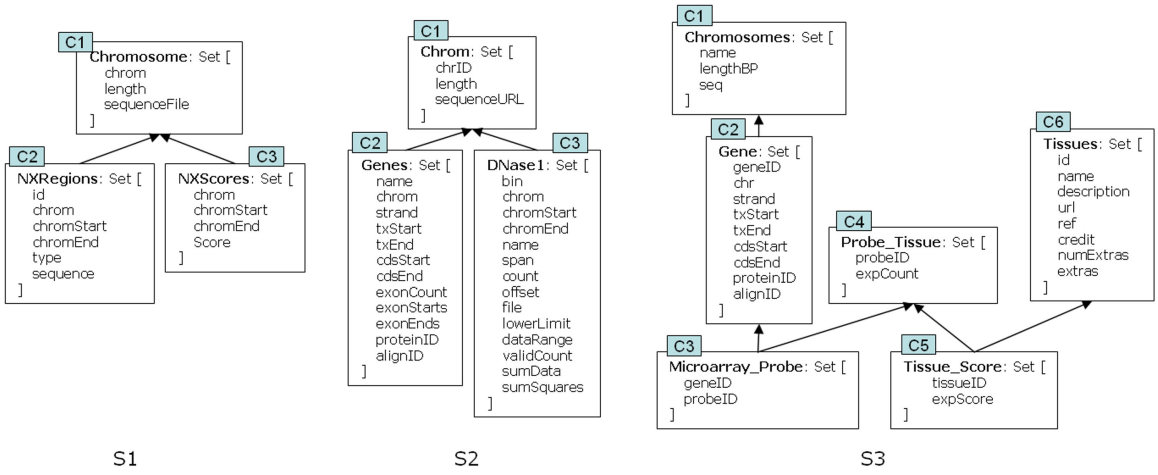

Figure 4.2: Input concepts corresponding to the source schemas shown in fig. 4.1.

	S2.C1		S2.C2		S2.C3	
S1.C1	(1,1)	**X_0=1**	(0.33,0.08)	**X_{12}=0.08**	(0.33,0.07)	**X_{16}=0.07**
S1.C2	(0.17,0.33)	**X_6=0.17**	(0.17,0.08)	**X_{13}=0.08**	(0.17,0.07)	**X_{17}=0.07**
S1.C3	(0.25,0.33)	**X_4=0.25**	(0.25,0.08)	**X_{14}=0.08**	(0.25,0.07)	**X_{18}=0.07**

(a)

	S3.C1		S3.C2	
S1.C1	(1,1)	**X_1=1**	(0.33,0.11)	**X_8=0.11**
S1.C2	(0.17,0.33)	**X_7=0.17**	(0.17,0.11)	**X_9=0.11**
S1.C3	(0.25,0.33)	**X_5=0.25**	(0.25,0.11)	**X_{10}=0.11**

(b)

	S3.C1		S3.C2	
S2.C1	(1,1)	**X_2=1**	(0.33,0.11)	**X_{11}=0.11**
S2.C2	(0.08,0.33)	**X_{15}=0.08**	(0.75,1)	**X_3=0.75**
S2.C3	(0.07,0.33)	**X_{19}=0.07**	(0.07,0.11)	**X_{20}=0.07**

(c)

Figure 4.3: Distances between the input concepts.

X_{20}	X_{19}	X_{18}	X_{17}	X_{16}	X_{15}	X_{14}	X_{13}	X_{12}	X_{11}	X_{10}	X_9	X_8	X_7	X_6	X_5	X_4	X_3	X_2	X_1	X_0	
0.07	0.07	0.07	0.07	0.07	0.08	0.08	0.08	0.08	0.11	0.11	0.11	0.11	0.17	0.17	0.25	0.25	0.75	1	1	1	$\hat{S}$
0.93	0.93	0.93	0.93	0.93	0.92	0.92	0.92	0.92	0.89	0.89	0.89	0.89	0.83	0.83	0.75	0.75	0.25	0	0	0	$\hat{D}$
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	A_1
0.86	0.86	0.86	0.86	0.86	0.84	0.84	0.84	0.84	0.78	0.78	0.78	0.78	0.66	0.66	0.5	0.5	0.5	1	1	1	Δf
X_2	X_1	X_0	X_{20}	X_{19}	X_{18}	X_{17}	X_{16}	X_{15}	X_{14}	X_{13}	X_{12}	X_{11}	X_{10}	X_9	X_8	X_7	X_6	X_5	X_4	X_3	Map
1	1	1	0.86	0.86	0.86	0.86	0.86	0.84	0.84	0.84	0.84	0.78	0.78	0.78	0.78	0.66	0.66	0.5	0.5	0.5	Δf_s
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	$K=2$
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	$K=3$
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	$K=4$
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	$K=5$

Figure 4.4: Enumeration algorithm to find the top-k schemas.

a vector $\Delta f = [\Delta f_{n-1}...\Delta f_0]$, where $\Delta f_i = |\hat{S}_i - \hat{D}_i|$. For each i, Δf_i represents the increase in cost - w.r.t. $cost(\mathbb{A}_1)$ - if the bit i (i.e., variable X_i) in $\mathbb{A}_1$ were to be flipped (from the current value to its complement). We sort the Δf vector in increasing order, and denote the new sorted vector as Δf_s. At the same time, we keep track in a Map vector of the new positions of the variables X_i after sorting. We next proceed to explore incremental modifications of the original assignment $\mathbb{A}_1$ in order to explore the next best assignments (the 2nd best, the 3rd best, and so on). The incremental modifications are based on the sorted vector Δf_s of cost increases. Specifically, if we look at Figure 4.4, it can be seen that the 2nd best assignment can be obtained by just flipping the bit X_3, since this will give the least cost increase (according to Δf_s). Next, to compute the 3rd best assignment, we need to change the variable with the next cost increase (i.e., X_4) and leave X_3 unflipped (relative to $\mathbb{A}_1$). The proposed algorithm is Section 3.5 presents a systematic approach for enumerating the top k assignments which is polynomial in k. The algorithm iteratively outputs the top-k assignments in increasing cost in the form of bit vectors $\mathbb{F} = [f_{n-1}...f_0]$. Recall that, Δf_{s_i} is the increase in cost associated with setting bit f_i to 1. Also, recall that these $\mathbb{F}$ bits are all 0 for the first assignment $\mathbb{A}_1$.

Figure 4.5 shows the resulting top 5 assignments. These assignments can be used to generate the corresponding combined concept graphs and hence the corresponding inte-

X_{20}	X_{19}	X_{18}	X_{17}	X_{16}	X_{15}	X_{14}	X_{13}	X_{12}	X_{11}	X_{10}	X_9	X_8	X_7	X_6	X_5	X_4	X_3	X_2	X_1	X_0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	A_1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	A_2
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	A_3
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1	A_4
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	1	1	A_5

Figure 4.5: Top-5 assignments (integrated schemas).

grated schemas. We ran the top-k algorithm to generate the top 64 and 128 schemas and calculated the time taken, the table in Figure 4.6 shows the results.

Attribute Corresp.	Source Concepts	Concept Corresp.	Time (ms) Top-64	Time (ms) Top-128
36	12	21	1.47	1.73

Figure 4.6: Results for generating the top-64 and the top-128 schemas.

After evaluating the top 10 integrated schemas, we have decided to use the The first (top) integrated schema. The top assignment in Figure 4.5 was used to generate the combined concept graph shown in Figure 4.7(a). The graph has nine concepts, we can see that the algorithm correctly merged the three different alternatives for the $Chromosme$ concept, in addition to merging the $Gene$ concept in two of the input graphs ($S2$ and $S3$) into a single concept in the combined graph. The integrated schema corresponding to the concepts graph is also shown in Figure 4.7(b). This integrated schema was used to facilitate the data manipulation capabilities required in this study.

4.4 Methods

4.4.1 Locating Nucleosome Exclusion Regions

We used a slightly modified version of nucleosome exclusion patterns identified in [89], which in turn were based on experimental data from a variety of sources [119, 143, 135]. These sequences were used to locate nucleosome exclusion regions (NXRs) throughout the human genome:

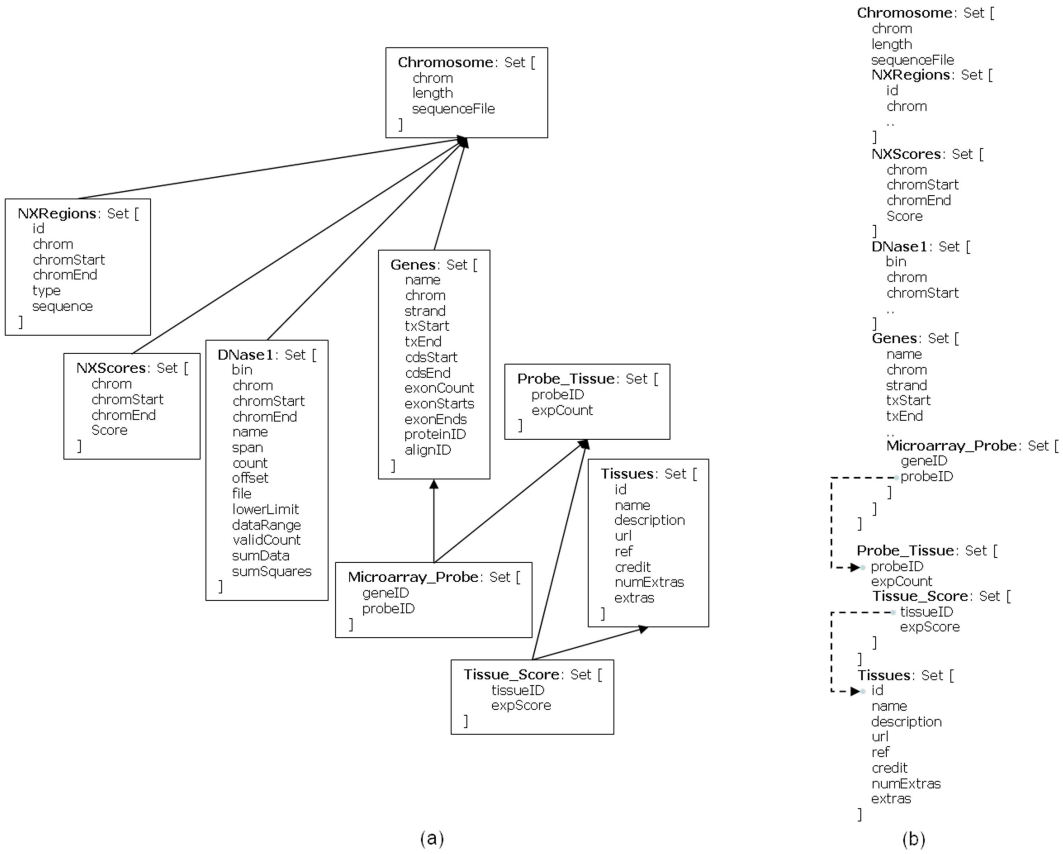

Figure 4.7: (a) Integrated concepts graph corresponding to the best (first) assignment, and (b) Equivalent integrated schema.

$[(G/C)_3N_2]_{\geq 3}$; e.g.: GGCAACGCTTGGGTA

$A_{\geq 10}(= T_{\geq 10})$; e.g.: AAAAAAAAAA, TTTTTTTTTT

It should be noted that our algorithm did not include sequences that had a weaker tendency to exclude nucleosomes, or that were rare on a genome-wide level, such as TGGA repeats [29]. This is because nucleosomes are known to slide [61] and we did not want to annotate a weak signal in case nucleosomes could slide into that particular region. Having said that, we do intend to update the annotations on the supporting online website when other strong exclusion sequences are reported and verified.

The hg18 (March 2006) human genome build was downloaded from the UCSC Genome Browser, and scanned base by base for NXRs. NXRs were annotated, and overlapping patterns were merged into one contiguous region in the final annotation. The annotations were compiled into a well supported exchange format for Feature description, GFF.

Nucleosome Exclusion Score Calculation

The nucleosome exclusion score (NXScore) measures the tendency of a specific DNA region to exclude nucleosomes. In order to have a continuous score across query sequences of variable length, the NXScore for each single base pair was calculated relative to a 147 base pair window, defined as the *neighborhood* of a particular nucleotide, centered at that nucleotide. The results per nucleotide are used to calculate the NXScore for any given region, as shown below:

- **NXScore calculation for a single base pair:** Calculating the NXScore per nucleotide depends on the density of NXRs in the 147 bps neighborhood of that nucleotide, however to fine tune our score calculation we specifically evaluated the weighted density of NXRs in the neighborhood. The idea behind the weighted density is to assign higher weights to NXRs close to the base pair under calculation than

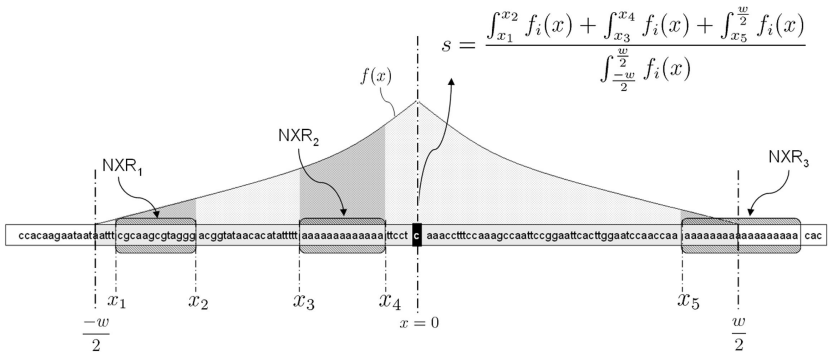

Figure 4.8: The NXScore at a nucleotide position depends on the weighted density of Nucleosome Exclusion Regions in the neighborhood surrounding the position.

distant NXRs. Figure 4.8 illustrates how this calculation is performed, the calculation is performed for the nucleotide at position $x = 0$ and $NXR_{1,2,3}$ are example exclusion regions, while $f_i(x)$ is the weighting function. We used a simple linearly decreasing weighting function, after finding that other functions yielded similar results, and maintaining that our main concern was identifying the peaks rather than the rate of change of the scores. For example, the score for a nucleotide whose 147 bp neighborhood contains one NXR of length x and located at either end of the neighborhood should be less than the score for a nucleotide whose 147 bp window contains one NXR of the same length x but at the center of the window (*i.e.*, surrounding the nucleotide in question).

NXScores can take the values 0 to 1 inclusive, such that if a nucleotide is centered in a neighborhood that is full of NXRs, then its NXScore will be equal to one. On the other hand, if the neighborhood is free from NXRs then the NXScore for that nucleotide will be zero. Figure 4.17 illustrates an example of NXRs and NXScores of a particular gene, chosen from chromosome 21. For display purposes, the NXScores in this figure and the rest of the figures in the study were scaled up to span the range from 0 to 1000 inclusive.

- **NXScore calculation for a sequence:** Having defined the NXScores for single nu-

cleotides, and given a DNA sequence of length n bp, its NXScore S_n is defined as the average NXScores for the n bp that make up that sequence. This can be represented by the formula: $S_n = \frac{1}{n}\sum_{i=1}^{n} s_i$, where s_i is the NXScore for bp i, and the summation is over the n bp.

NXScores annotations for the whole human genome are available in wiggle format [5] from the additional files.

4.4.2 Tissue Specificity Measures

A number of methods, based on microarray gene expression datasets, have been proposed for measuring the tissue specificity of gene use. Despite the inherent limitations of comparing microarray datasets, some methods have been able to describe trends in tissue specificity. In [120], the effectiveness of using Shannon entropy was demonstrated for ranking genes according to their tissue specificity, from narrow or tissue-specific expression, to wide or ubiquitous expression. Shannon entropy was used and updated in [77]. Earlier, a method derived from Akaike's information criterion, which was originally developed to detect outliers in a data set, was applied in [76], and was used to rank genes according to their tissue specificity.

Using the GNF-SymAtlas [3] gene expression dataset [133], we categorized known genes according to their tissue specificity levels, and investigated their possible correlation to NXScores.

Algorithms Used: We propose a new and efficient technique for ranking genes according to their tissue specificity, based on Grubbs' outliers test [60]. To validate our results we also used the previously published ranking mechanism utilizing Shannon's entropy. Both techniques gave almost the same results, verifying that this use of Grubbs' test is valid. It should be noted that the proposed technique in this study has the advantage of being able to detect both up-regulated and down-regulated genes in a microarray data set. We defined up-regulated genes as those that are expressed at a significantly high level in a limited group

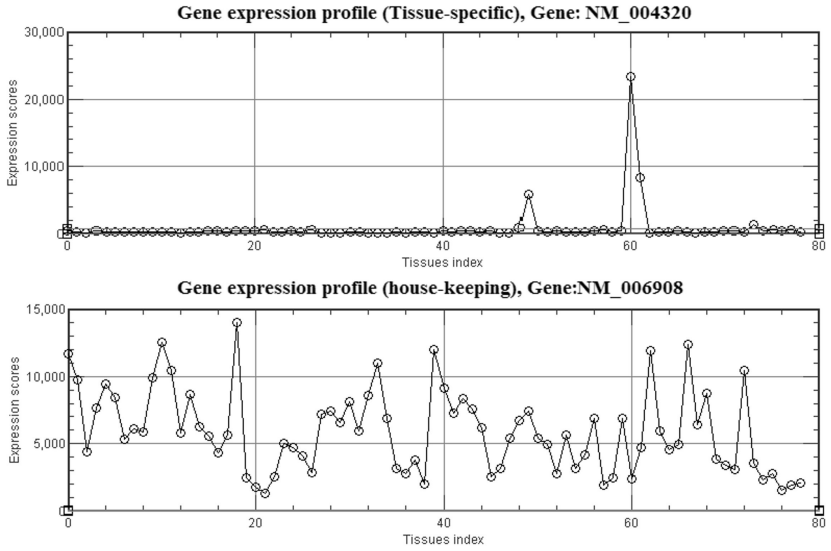

Figure 4.9: Tissue-specific and wide-use genes. Examples of the expression profile for a) a tissue-specific gene, NM_004320, and b) a wide-use gene, NM_152422.

of tissues compared to their expression in other tissues, and down-regulated genes as those that are expressed at significantly lower levels in a limited group of tissues compared to mid- to high- expression in other tissues. Even though for this particular study we only used up-regulated genes, the applicability of this method is valid for other data sets.

Figure 4.9 shows examples of the expression profiles of a tissue-specific gene (NM_004320, ATP2A1) and a gene that has a wide tissue distribution (NM_006908, RAC1). The $x-axis$ represents the tissues index, while the $y-axis$ depicts the expression scores. Strictly speaking, microarray data are not quantitative measures of expression levels, but they do give some indication of the trends. These values were used in the tissue specificity ranking calculations, detailed as follows:

- Grubbs' outliers test: The Grubbs' test [60], also known as the maximum normalized residual test, can be used to test for outliers in a univariate data set. Given the expression profile of a gene, the Grubbs' test G can be calculated as $G = \frac{max(w_t - \bar{w})}{std}$, where, $std = \sqrt{\frac{1}{n-1}\sum_{t=1}^{n}(w_t - \bar{w})^2}$, $\bar{w} = \frac{\sum_{t=1}^{n} w_t}{n}$, w_t is the expression score for tissue t, std is the standard deviation for the expression profile, and $\bar{w}$ is the mean

expression score. The more specific the gene, the higher the G value, and vice versa. While this formula for G identifies up-regulated genes, replacing max with min can identify down-regulated genes.

- Shannon's entropy: The concept of Shannon's entropy [123] has a central role in information theory, and is sometimes referred to as the measure of uncertainty. The entropy of a random variable is defined in terms of its probability distribution, and has been shown to be a good measure of randomness or uncertainty. The entropy is maximum when the variable is uniformly distributed, *i.e.*, it exhibits the highest uncertainty. Given a gene expression profile similar to those in Figure 4.17, the Shannon entropy (H) can be calculated as $H = \sum_{t=1}^{n} p_t.log_2(p_t)$, and, $p_t = \frac{w_t}{\sum_{t=1}^{n} w_t}$, where w_t represents the expression score for tissue t, and p_t is calculated by normalizing this value relative to the sum of expression scores for all tissues. The more specific the gene, the less its entropy, and vice versa.

4.5 Applications

4.5.1 Nucleosome Exclusion Landscape

First we constructed a whole genome landscape of nucleosome exclusion regions and calculated their exclusion scores. The results were compiled as GFF [1] and Wiggle [5] files for each of the human chromosomes, and are made available through the links provided in Additional Materials. This data is being made publicly available by the UCSC Genome Browser under their Custom Tracks Page [4].

Immediately obvious from the data is the fact that NXScores increase significantly at and around the transcriptional start sites (TSSs) of genes (Figure 4.10). This confirms previous observations that, regardless of how many nucleosomes there may be in a given promoter region, nucleosomes are preferentially excluded from the immediate area where

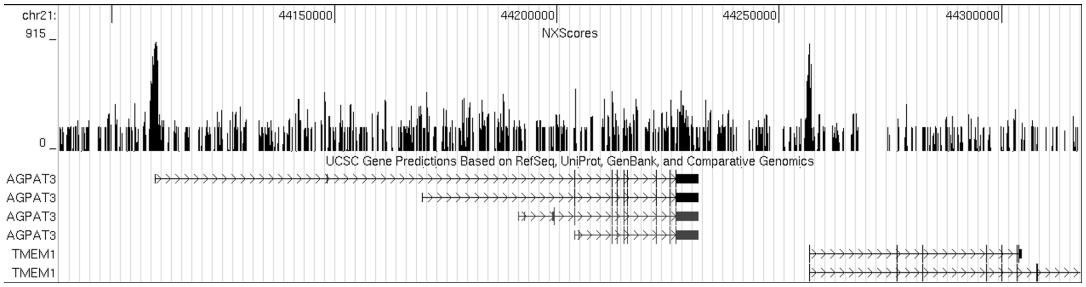

Figure 4.10: NXScore peaks around TSSs. An example of NXScore peaks around the transcriptional start site of genes. Shown above are the NXScores for two neighboring genes on chromosome 21. The figure was prepared by uploading NXScore results as a Custom Track on the UCSC Genome Browser, and taking a snapshot with the Known Genes track.

the transcriptional machinery needs easy access to the DNA [89]. The sections below highlight other observations and correlations we found.

4.5.2 Correlation with Gene Density

We observed a genome-wide correlation between NXScores and gene density, such that gene-rich areas have high NXScores (Figure 4.11). To validate this observation, we calculated the mean NXScore for each of the ENCODE regions [7] (Human Genome, UCSC Release hg18). We then counted the number of RefSeq genes in each region, and normalized that number by the size of the corresponding ENCODE region. Figure 4.11b shows the mean NXScore and the density of gene number for each ENCODE region. The data sets exhibit a strong positive correlation ($r = +0.71$) based on a Pearson product-moment correlation coefficient. This confirms the observation that gene-rich areas have high NXScores. Figure 4.11a illustrates this trend using chromosome 20, similar figures for all the human chromosomes are available in the supplementary data files.

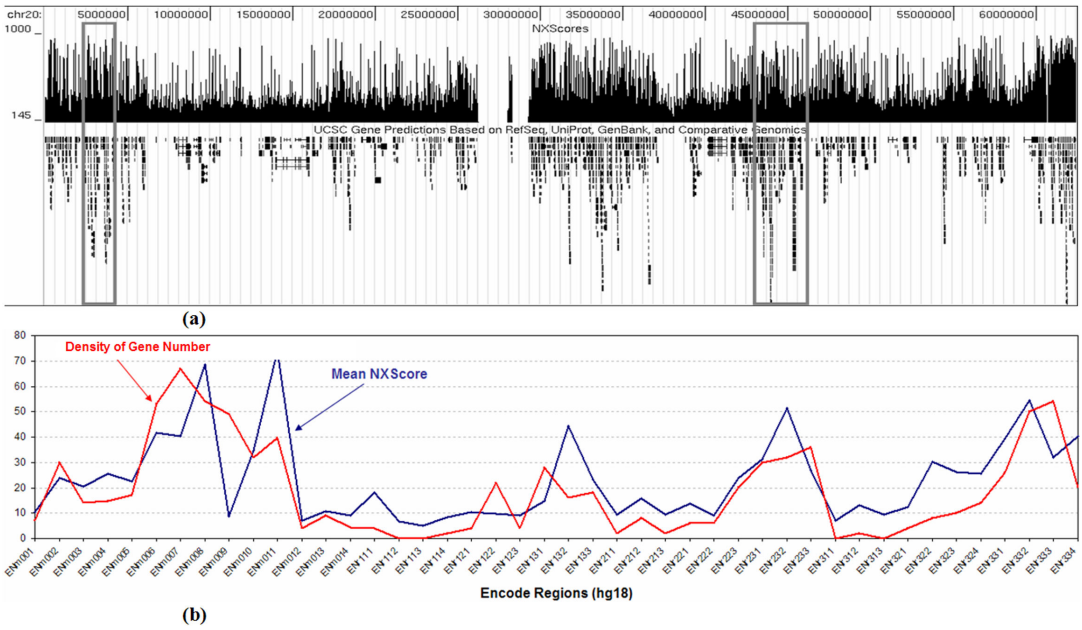

Figure 4.11: Correlation between NXScores and gene density. (a) Chromosome 20 is shown as an example, where the empty area in the middle is the centromere, and the boxes highlight two examples of gene-rich areas with high NXScores. The figure was prepared by uploading NXScore results as a Custom Track on the UCSC Genome Browser, and taking a snapshot with the Known Genes track. (b) The calculated correlation between NXScores and the density of gene number for the ENCODE regions of hg18.

4.5.3 Correlation with Tissue Specificity

We obtained the gene expression profiles of 19055 genes and developed a new method for ranking the tissue specificity of those genes. The available SymAtlas "tissue list" includes 79 cell types, tissues and organs, which makes it difficult to classify genes categorically into tissue specific groups. Furthermore, genes that have been classified as tissue specific by other researchers were often expressed equally in three or four different tissues. In order to overcome this problem, we refer to genes as having a wide tissue distribution if they are expressed at relatively equal levels in five tissues or more, and as having a narrow distribution if they are expressed at relatively equal levels in only one or two tissues. To follow this idea through, we needed a method of ranking genes according to their tissue distribution, so that we could correlate this with NXScores.

The RefSeq-annotated transcriptional start site (TSS) was used to identify the promoter region of each gene, and NXScores were calculated for the region TSS-1500 to TSS$+500$. The resulting values were used to sort the 19055 genes in ascending order (*i.e.*, from no nucleosome exclusion to complete nucleosome exclusion). The sorted list was then divided into n groups. The mean tissue specificity for each of these groups was calculated using a method we developed based on Grubbs' test [60], and we validated this method using an already established method for ranking tissue specificity based on Shannon entropy [123].

Grouping genes facilitated the inspection of the general trends among gene groups while filtering noise and extraneous behavior that maybe associated with specific limited number of genes (within the group). Hence, the number of groups n served as a zooming parameter for inspecting and visualizing such trends. Figure 4.12 shows the results for $n = 10$, illustrating the correlation between the tissue specificity of gene expression and NXScores. To provide a closer inspection of these trends, figures are made available in Additional Materials for the results of groups of $n = 5$ (zoom out), $n = 20$ (zoom in), and $n = 40$ (higher zoom in). The results show that previous localized findings [57, 89] are valid on a whole genome level. There is a direct correlation between the NXScores in the

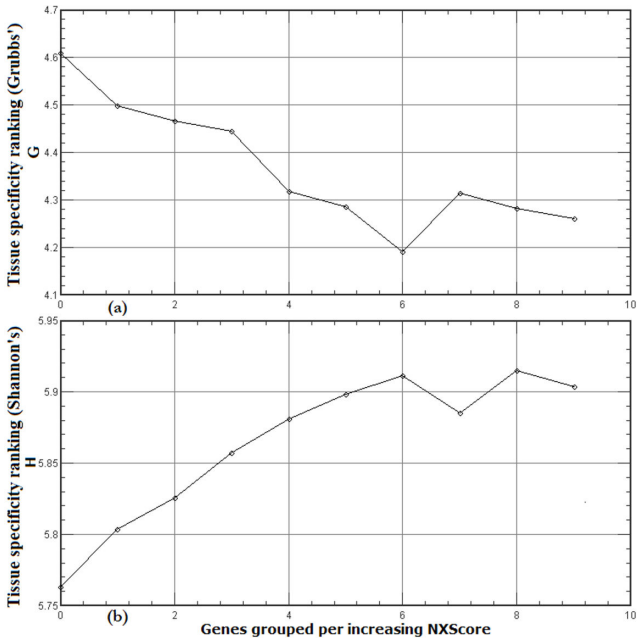

Figure 4.12: Correlation between NXScores and tissue specificity. The 19055 genes were arranged into 10 groups (0-9) of increasing NXScores for the -1500 to +500 promoter regions, and the mean tissue specificity level for each band was calculated using a new method based on Grubbs' test (a) and validated using a previously known method based on Shannon's entropy (b).

promoter region and tissue specificity. The higher the NXScore of a promoter sequence, the less likely it is to include a nucleosome, and the less tissue specific the associated gene is. Given the complexity of transcriptional regulation in the eukaryotic system, there may be a few exceptions to this, but the genome-wide trend is clearly observed from our results. One could deduce from this that the transcriptional machinery has relatively unimpeded access to the TSSs of widely expressed genes. It is expected that the types of transcription factors that switch on widely expressed genes are generally not those that can tolerate the DNA being wound around a nucleosome.

To take a closer look at the promoter region, the genes were sorted according to their measure of tissue specificity, then grouped into three groups; group 1 constitutes the top 10% tissue specific genes, group 2 constitutes the top 20% tissue specific genes, while group 3 constitutes the whole collection of 19055 genes under inspection. For every gene,

the NXScore for each base pair in the promoter region was calculated and then averaged separately for genes of each group. The objective was to inspect promoter NXScores profiles among genes with varying tissue specificity levels. Again, note that groups and averaging were used to inspect general trends while filtering noise and extraneous behavior that may be associated with a limited number of genes within each group. The results (Figure 4.13) show that the NXScore peak is approximately 30 bases upstream of the TSS, and that there is a shoulder immediately downstream from the TSS, extending approximately 250 bases into the gene. There is thus a tendency for the region surrounding the TSS to be nucleosome-free, regardless of whether the gene is widely or narrowly expressed. This presumably helps maintain the momentum of the transcriptional machinery as it moves from the TSS through the first part of the gene. After that point, there is a significant decrease in mean NXScore before it levels out, implying that the remainder of the gene is more likely occupied by nucleosomes. This is in agreement with ENCODE findings that regulatory sequences that surround transcription start sites are symmetrically distributed [8]. Our results indicate that there is a gradually increasing tendency for the promoter to be nucleosome-free the closer one gets to the TSS (Figures 4.10 and 4.13). We used the RefSeq gene-annotations of transcriptional start sites (RefSeq-TSSs), and found the average NXScore to peak about 30bp upstream from the RefSeq-TSS. However, we also found that the RefSeq-TSSs themselves are often 20-40 bp downstream from the TSSs determined by experimental methods [136, 30]. Therefore the peak of nucleosome exclusion seen in our results appears, on average, to be centered on the transcriptional start site. This is in agreement with the findings of [104], who provided experimental evidence that the region around the TSS in humans was relatively nucleosome-free.

Figure 4.13 highlights that all 19055 genes follow the trend explained above, and that the more tissue specific groupings follow that trend but with lower NXScore peaks. The top 10% most tissue specific genes have the lowest NXScore peak, meaning that even though their TSS region is depleted of nucleosomes, there are more exceptions to that trend

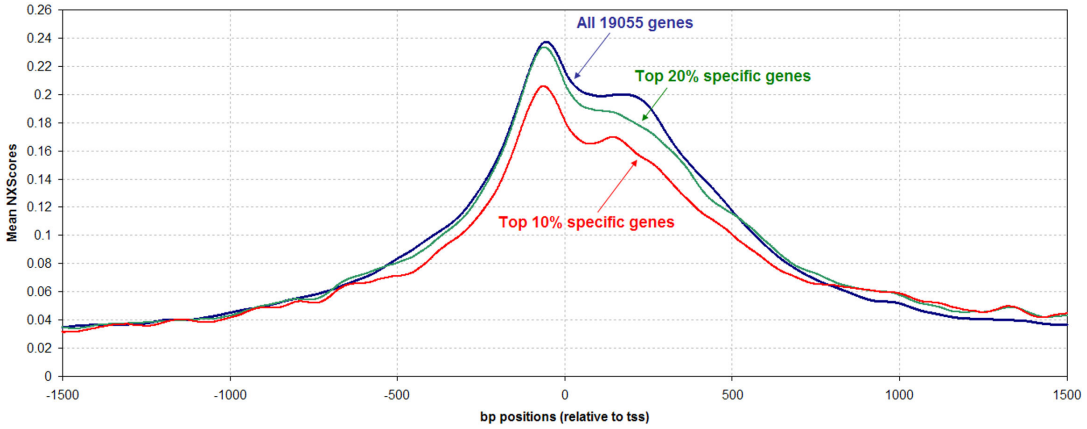

Figure 4.13: Mean NXScores for promoter base pair positions -1500 to +1500. The red line represents the mean NXScore across the top 10% most tissue specific genes, *i.e.* those with the narrowest tissue distribution, the green line represents the mean across the top 20%, *i.e.* a grouping of slightly wider distribution genes, and the blue line represents the mean across all the 19055 genes sampled.

in this group than there are across all the genes. This is in agreement with the previous conclusion that the more tissue-specific a gene is, the more likely it is to have nucleosomes on its promoter. The differences in NXScore peak value observed here suggest that with gradually increasing tissue specificity, nucleosome binding to promoter regions plays an increasingly important role in gene regulation.

4.5.4 Correlation with Gene Expression Level

NXScores for each gene were calculated from the RefSeq-annotated TSS to the RefSeq-annotated 3'UTR end of the gene, including all exons and introns. Then the median expression level was calculated for each gene using the SymAtlas gene expression profiles [133]. We calculated the median in order to filter very high or very low expression levels that may be associated with specific tissues, since our objective for this analysis was to capture expression levels across each gene irrespective of tissue specificity. The genes were then sorted according to increasing NXScore, the sorted list was equally divided into 5 groups, and the mean expression level was calculated for each group. This grouping and

the calculations undertaken were used to inspect general trends while filtering noise that may be associated with a limited number of genes within each group.

The data show that gene expression level is positively correlated with high NXScore (Figure 4.14a), and that expression level drops with very high NXScores. This can be clearly seen if we zoom in slightly and divide the data set into 10 groups, as illustrated in Figure 4.14b. In other words, the peak in expression level is around moderate NXScores: expression is lower when there are a lot of nucleosomes present (lower NXScore), and it is also lower when there are hardly any nucleosomes present (high NXScore). NXScores are calculated using G/C-rich sequence patterns [89], and G-C pairing involves three hydrogen bonds, whereas A-T pairing involves only two, which allows us to speculate that the lower expression levels of genes with the very high NXScores may reflect slower movement of the transcription machinery through regions of very high G-C content.

4.5.5 Experimental Validation

Thus far all our observations have been *in silico*. To validate our annotations, we compared our scores to conserved nucleosome locations that have been reported in recent studies [104, 85]. The study in [104] reported nucleosome occupancy on the promoter regions of several human genes, and we looked at the NXScores of those exact sequences. For further validation, we ran the NXScores algorithm on selected regions of the *Saccharomyces cerevisiae* genome, namely those used in [85], to report experimentally verified nucleosome positions.

It is evident from Figure 4.15 (more graphs are available in Additional Materials) that although our nucleosome exclusion predictions and the experimentally verified nucleosome positions correlate well, they do not correlate exactly. In some cases, NXScores did not predict nucleosome depletion in a region where no nucleosomes were found. The results constitute a 7% false negative error margin, and for this we have two possible explanations. Firstly, we suggest that the sequences not picked up by NXScores may be regions to

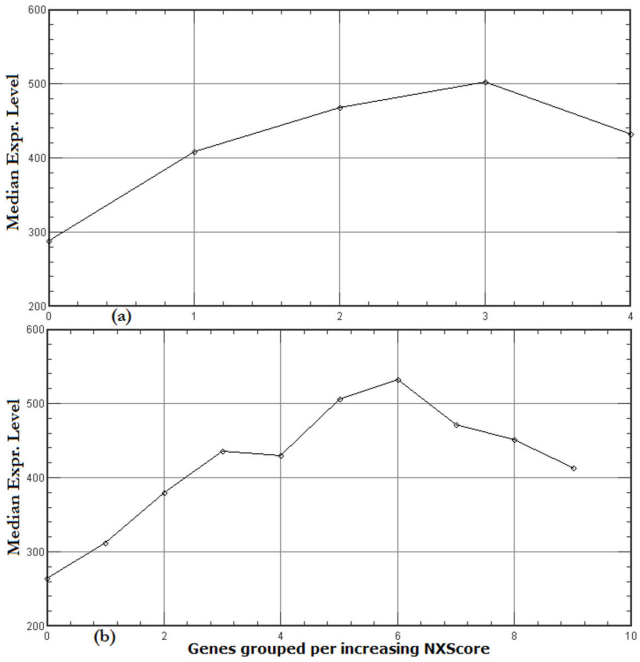

Figure 4.14: Correlation between NXScores and gene expression levels. The 19055 genes were first ranked according to increasing NXScore for the whole gene (TSS to end of 3'UTR), and the sorted list was divided into (a) 5 and (b) 10 groups. The mean value of the median expression level for the genes in each group was plotted. The graph shows that NXScores are positively correlated with gene expression level except at the very high NXScores, where expression levels decrease.

which nucleosomes slide according to the transcriptional activity state of the promoter at any given time. More importantly, however, these discrepancies highlight the fact that we were stringent in our choice of nucleosome exclusion sequences for our algorithm. We did not use weaker nucleosome exclusion sequences that have been reported in the literature because we wanted to have a certain level of confidence in predicting where nucleosomes will not bind, and assume that they may, at some developmental or physiological state, bind on the weaker exclusion signals [51, 113].

The study in [104] calculated the average $log_2(Cy5/Cy3)$ data of 57 MITF-bound promoters in the human genome. We compared these results with our calculated average NXScores promoter profile for the 19055 genes under inspection (Figure 4.15c), and obtained a medium-to-strong negative correlation ($r = -0.47$ based on Pearson product-moment correlation coefficient). This correlation is satisfactory keeping in mind that nucleosomes can slide according to the transcriptional activity of the promoter, and that our profile was calculated as a consensus promoter profile representing the 19055 genes, while the nucleosome positioning results were obtained using 57 MITF-bound promoters. In fact, overall, there were almost no examples where NXScores were high on areas that were experimentally shown to be occupied by nucleosomes.

4.5.6 Correlation with DNaseI Hypersensitive sites

As a final comparison, we looked at whether nucleosome exclusion scores would correlate with DNaseI hypersensitive sites (DHSs). It is known that nucleosome-free areas are more prone to digestion by DNaseI, and it was reported in [146] that ubiquitous DHSs, shared by 6 cell lines, were found near the transcriptional start sites of some genes, implying a wide usage of that gene, or at least of that promoter.

The study in [59] predicted the hydroxyl radical cleavage intensity on naked DNA for each nucleotide in the ENCODE regions. We downloaded this data for the whole set of hg18 ENCODE regions from the UCSC Genome Browser, and calculated the mean value

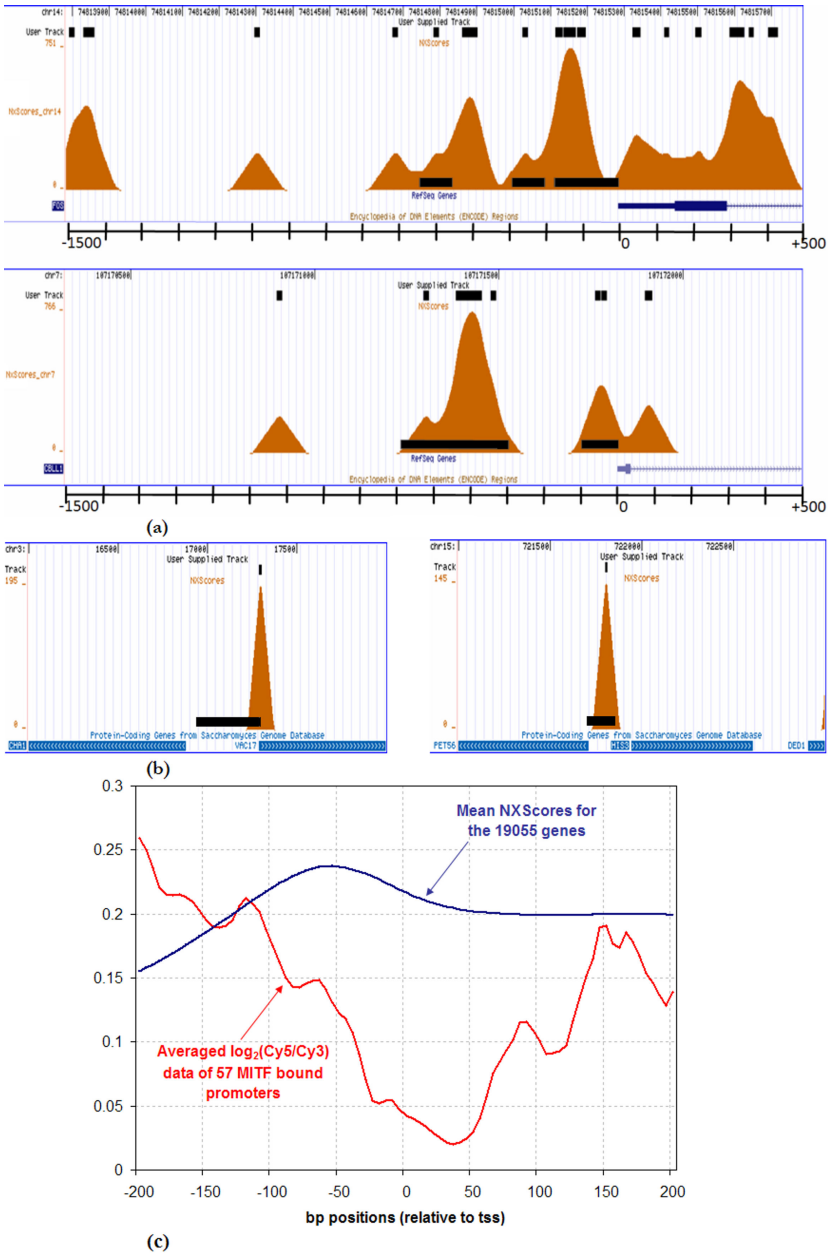

Figure 4.15: Correlation between NXScores and experimentally verified nucleosome exclusion regions. (a) The -1500 to +500 promoter region of the human genes FOS and CBLL1 are shown with the nucleosome positions from Ozsolak *et al.* (2007) denoted by black bars superimposed on the NXSensor graphics. The results and correlations of several other genes can be found in Additional Materials. (b) The promoter regions of the yeast benchmark genes CHA1 and HIS3 are shown with the nucleosome positions from Lee *et al.* (2007) denoted by black bars as above. The NXScore results were uploaded as a Custom Track on the UCSC Genome Browser, and a snapshot was taken with the human RefSeq genes track (a), or the protein coding genes track from yeast (b). (c) The correlation between the mean NXScores for the TSS-200 to TSS$+200$ promoter region of 19055 genes, and the calculated average $log_2(Cy5/Cy3)$ data of 57 MITF-bound promoters in the human genome.

of the predicted cleavage intensity for each region. The objective of this analysis was to investigate whether regions with high NXScores would have a high predicted cleavage intensity.

First, we calculate the NXScores for each region, and took the locations for NXScore peaks that had NXScores higher than p, where $p = \mu + (\tau \times s)$. μ and s are the mean and standard deviation of the NXScores across the region, respectively. τ is a parameter for determining the height of the calculated peaks, such that the higher the τ value, the higher the peak value and fewer the number of the peaks across the regions, and vice versa. Next, for each peak location, we calculated the mean predicted cleavage intensity of a 147 bps neighborhood centered at the peak, and we averaged these values for all peak locations in a specific region for a specific τ.

In this way we were able to show that the mean predicted cleavage intensity around the peaks is higher than the mean intensity across the whole region, thus proving that regions with a high NXScore also have a high cleavage intensity.

To further investigate this, we varied τ from 3 to 9 and reported the results for each τ. As expected, the higher the τ, the higher the mean predicted cleavage intensity. Figure 4.16a shows the ratio between the average calculated cleavage intensity around the peaks and the average cleavage intensity for that whole region, for all ENCODE regions at different τ values. When $\tau = 3$, the mean intensity increased by approximately 26%, and the intensity increased with increasing τ, reaching a 61% increase when $\tau = 9$. Figure 4.16b illustrates this correlation for ENCODE region ENr231. The table in Figure 4.17 shows a detailed account of these calculations for each ENCODE region.

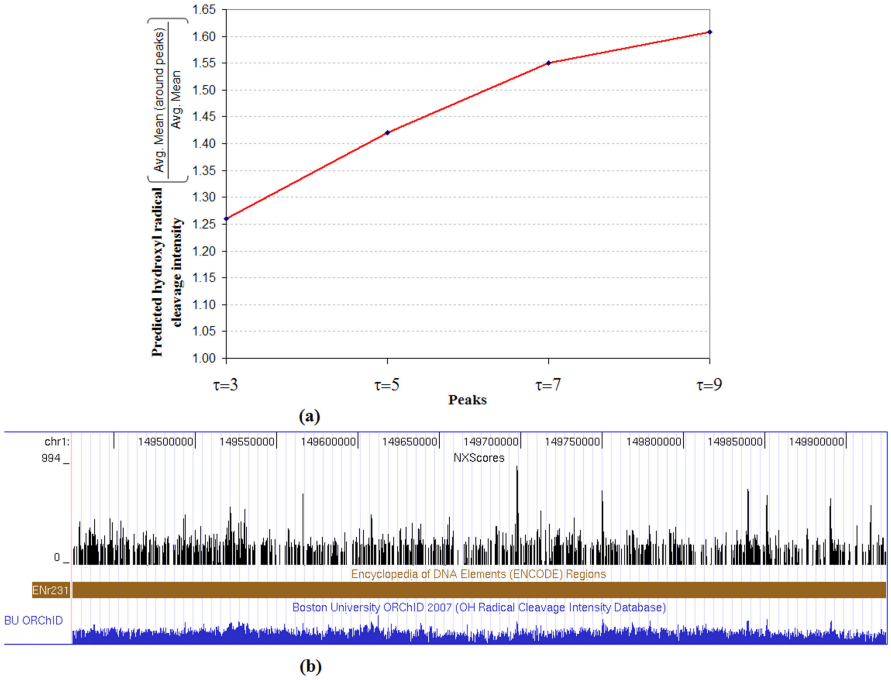

Figure 4.16: Correlation between NXScores and DNaseI hypersensitive sites. (a) This graph shows the ratio between the average calculated cleavage intensity around the peaks and the average cleavage intensity for that whole region, for all ENCODE regions at different τ values. (b) The tracks for NXScores and DNaseI hypersensitive sites are shown for ENCODE region ENr231 in a snapshot of a UCSC Genome Browser screen.

Encode Region	Size (~Mb)	Location			NXScores μ	NXScores s	Mean Cleavage Intensity for the whole region	Mean Cleavage Intensity around peaks (p >= μ + τ s)			
		chr	start (bp)	end (bp)				τ=3	τ=5	τ=7	τ=9
ENm001	1.9	chr7	115597757	117475182	10.561	46.215	0.239	0.298	0.341	0.426	0.494
ENm002	1	chr5	131284314	132284313	23.784	73.145	0.279	0.403	0.465	0.481	0.481
ENm003	0.5	chr11	115962316	116462315	20.495	59.257	0.266	0.323	0.349	0.412	0.433
ENm004	1.7	chr22	30133954	31833953	25.707	69.276	0.293	0.344	0.419	0.457	0.472
ENm005	1.7	chr21	32668237	34364221	22.365	66.892	0.276	0.343	0.426	0.468	0.482
ENm006	1.2	chrX	152767492	154063081	41.572	99.782	0.307	0.455	0.493	0.509	0.502
ENm007	1	chr19	59023585	60024460	40.286	90.656	0.318	0.419	0.471	0.492	0.556
ENm008	0.5	chr16	1	500000	68.723	128.570	0.347	0.476	0.497	0.590	N/A
ENm009	1	chr11	4730996	5732587	8.678	38.816	0.238	0.276	0.289	0.298	0.288
ENm010	0.5	chr7	26924046	27424045	34.681	93.856	0.276	0.448	0.488	0.515	0.477
ENm011	0.6	chr11	1699992	2306039	73.909	120.576	0.358	0.468	0.510	0.507	N/A
ENm012	1	chr7	113720369	114720368	7.197	34.668	0.223	0.249	0.266	0.280	0.357
ENm013	1.1	chr7	89621625	90736048	10.691	47.728	0.241	0.311	0.358	0.408	0.425
ENm014	1.2	chr7	125865892	127029088	9.007	44.933	0.229	0.310	0.367	0.427	0.479
ENr111	0.5	chr13	29418016	29918015	18.236	61.283	0.269	0.335	0.422	0.464	0.478
ENr112	0.5	chr2	51512209	52012208	6.795	33.546	0.212	0.239	0.270	0.284	0.303
ENr113	0.5	chr4	118466104	118966103	5.091	26.656	0.208	0.230	0.248	0.244	0.234
ENr114	0.5	chr10	55153819	55653818	8.264	36.039	0.211	0.257	0.271	0.287	0.361
ENr121	0.5	chr2	118011044	118511043	10.339	43.128	0.255	0.310	0.353	0.415	0.476
ENr122	0.5	chr18	59412301	59912300	9.643	42.295	0.243	0.302	0.338	0.368	0.434
ENr123	0.5	chr12	38626477	39126476	8.915	43.838	0.221	0.283	0.339	0.399	0.433
ENr131	0.5	chr2	234156564	234656627	14.939	52.295	0.281	0.345	0.365	0.382	0.386
ENr132	0.5	chr13	112338065	112838064	44.521	95.540	0.356	0.459	0.474	0.471	N/A
ENr133	0.5	chr21	39244467	39744466	23.290	69.686	0.269	0.355	0.431	0.471	0.482
ENr211	0.5	chr16	25780428	26280428	9.463	37.266	0.269	0.281	0.274	0.284	0.322
ENr212	0.5	chr5	141880151	142380150	15.908	51.509	0.284	0.313	0.345	0.404	0.437
ENr213	0.5	chr18	23719232	24219231	9.252	46.069	0.227	0.294	0.359	0.427	0.465
ENr221	0.5	chr5	55871007	56371006	13.804	56.612	0.250	0.332	0.438	0.479	0.486
ENr222	0.5	chr6	132218540	132718539	9.079	39.837	0.234	0.275	0.292	0.351	0.504
ENr223	0.5	chr6	73789953	74289952	23.898	64.889	0.258	0.319	0.375	0.455	0.470
ENr231	0.5	chr1	149424685	149924684	31.232	78.755	0.287	0.364	0.437	0.465	0.480
ENr232	0.5	chr9	130725123	131225122	51.480	100.348	0.342	0.443	0.496	0.528	N/A
ENr233	0.5	chr15	41520089	42020088	27.021	69.208	0.283	0.323	0.385	0.403	0.379
ENr311	0.5	chr14	52947076	53447075	6.978	32.159	0.237	0.264	0.279	0.258	0.249
ENr312	0.5	chr11	130604798	131104797	13.255	48.535	0.288	0.327	0.339	0.391	0.415
ENr313	0.5	chr16	60833950	61333949	9.312	39.058	0.228	0.249	0.253	0.275	0.304
ENr321	0.5	chr8	118882221	119382220	12.543	47.082	0.253	0.284	0.311	0.384	0.491
ENr322	0.5	chr14	98458224	98958223	30.247	79.229	0.295	0.394	0.436	0.462	0.489
ENr323	0.5	chr6	108371397	108871396	26.284	75.788	0.262	0.365	0.437	0.488	0.528
ENr324	0.5	chrX	122609996	123109995	25.457	65.643	0.256	0.294	0.378	0.424	0.445
ENr331	0.5	chr2	219985590	220485589	39.218	96.131	0.321	0.446	0.484	0.512	0.469
ENr332	0.5	chr11	63940889	64440888	54.382	107.313	0.339	0.454	0.490	0.493	N/A
ENr333	0.5	chr20	33304929	33804928	31.906	75.387	0.290	0.350	0.423	0.478	0.524
ENr334	0.5	chr6	41405895	41905894	40.257	96.330	0.317	0.426	0.476	0.489	0.523
						Average	0.271	0.342	0.385	0.421	0.436
						Ratios		1.260	1.420	1.550	1.608

Figure 4.17: Correlation between NXScores and DNaseI hypersensitive sites (calculations). This table shows the results and all intermediate calculations for the correlations between NXScores and DNaseI hypersensitive sites for all ENCODE regions of hg18.

Chapter 5

A Cloud Data Processing Abstraction Layer

Using the low-level MapReduce for general data processing tasks poses the problem of developing, maintaining and reusing custom low-level user code. In this chapter, we propose and describe a novel refined MapReduce model, MR-LEGOS, an explicit model for composing MapReduce constructs from simpler components, namely, "Maplets", "Reducelets" and optionally "Combinelets". Maplets and Reducelets are standard MapReduce constructs that can be composed to define aggregated constructs describing the problem semantics. We'll show that by using the proposed model, complex problem semantics can be defined in an easier and more efficient way. The model is analogous to LEGO bricks that can be connected in various configurations to construct different shapes. Having a collection of these standard and reusable predefined bricks, helps define complex processing tasks in an efficient, scalable and easily maintainable way. We present the design details, usage scenarios, performance analysis and highlight the main features of MR-LEGOS.

5.1 MapReduce LEGOS Job Model

As previously discussed, in a conventional MapReduce job, the programmer basically provides two functions/classes; a Mapper and a Reducer. The Mapper and Reducer are used to describe the MapReduce job semantics. MR-LEGOS refines such composition where a set of Mappers (Maplets) and Reducers (Reducelets) can be used to compose the MapRe-

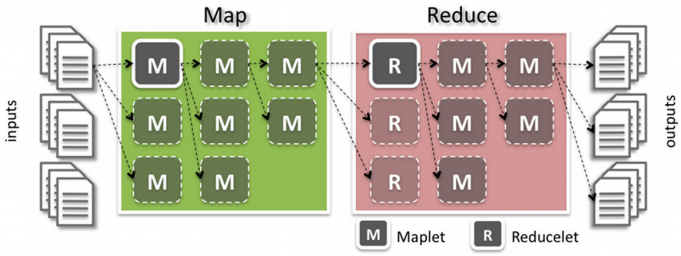

Figure 5.1: Internal design of the refined MR-LEGOS job definition. The Map is composed of a collection of Maplets, while the Reduce is composed of a parallel array of Reduclets followed by a collection of Maplets. The MR-LEGOS job definition describes how these Maplets/Reducelets are connected within the job.

duce job. MR-LEGOS also provides the capability of grouping input and outputs into logical units. Accordingly, using MR-LEGOS, the programmer provides a list of logically grouped inputs and outputs and a set of Maplets and Reducelets that are used in the job composition. Additionally, the programmer provides a definition of how these Maplets and Reducelets are interconnected and how they are connected to the inputs/outputs.

Figure 5.1 shows the design details. The job has one or more logical inputs. The Mapper is composed from one or more Maplets, a MR-LEGOS micro-workflow (the dotted arrows in the diagram) describes how inputs are connected to Maplets and how such Maplets are interconnected. The Reducer is composed from a array of Reducelets optionally followed by a set of Maplets; the MR-LEGOS micro-workflow again describes the connections within the Reducer and to a set of one or more logically grouped outputs.

5.1.1 Composing Maplets and Reducelets

We briefly highlight in this section the main features that enable MR-LEGOS to provide such refined MapReduce job definition. Figure 5.2 depicts four main composition cases.

(a) One or more Mappers connected in series: This case is illustrated in Figure 5.2(a), where a set of Maplets are connected in series and the output from one Maplet is fed as input to the next Maplet. This sequence of Maplets is equivalent to a single compound Mapper with a Map function expressed as $map_c(k1, v1) = map_n(map_{n-1}(...map_1(k1, v1)))$,

where map_c represents the compound Map function, and $map_{1..n}$ represent the individual Map functions for each Maplet. This compound Mapper preserves the Map function associated type presented in Eq. (1.1).

(b) One or more Mappers connected in parallel: This case is illustrated in Figure 5.2(b). We can see that such an arrangement is also equivalent to a single compound Mapper if such Mapper has a demultiplexing module at its input and a multiplexing module at its output; the demultiplexing module decides which key/value pairs are forwarded to which Maplet, and the multiplexing module combines the outputs key/value pairs from the individual Maplets into a single output stream of key/value pairs. The compound Map function is expressed as $map_c(k1, v1) = \sum_{i \in S} map_i(k1, v1)$, where $S \subseteq N$, and N is the set of all Maplets encompassed by this compound Mapper. This compound Mapper also preserves the Map function associated type presented in Eq. (1.1).

(c) A Reducer followed by zero or more Mappers connected in series: Such an arrangement is equivalent to a single compound Reducer composed from a single Reducer and each output key/value pair from such Reducer is piped through a set of Mappers connected in series. This case is depicted in Figure 5.2(c). The compound Reduce function is expressed as $reduce_c(k1, list(v1)) = map_n(map_{n-1}(...reduce(k1, list(v1))))$. This compound Reducer preserves the Reduce function associated type presented in Eq. (1.2).

(d) One or more Reducers connected in parallel: This arrangement is also equivalent to a single compound Reducer if such Reducer has a demultiplexing module at its input and a multiplexing module at its output; the demultiplexing module decides which key/value pairs are forwarded to which Reducelet, and the multiplexing module combines the outputs key/value pairs from the individual Reducelets into a single output stream of key/value pairs. This case is depicted in Figure 5.2(d). The compound Reduce function is expressed as $reduce(k1, list(v1)) = \sum_{i \in S} reduce_i(k1, list(v1))$, where $S \subseteq N$, and N is the set of all Reducelets encompassed by this compound Reducer. This compound Reducer also preserves the Reduce function associated type presented in Eq. (1.2).

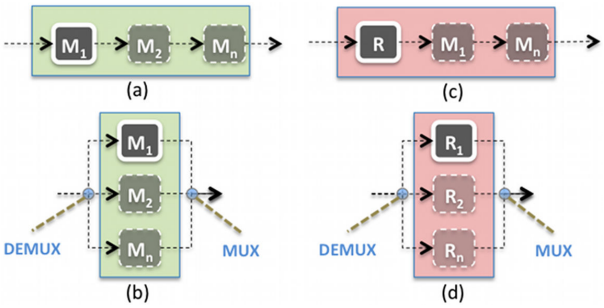

Figure 5.2: Different primitive combinations for composing Maplets and Reducelets. (a) A set of Maplets connected in series, (b) A set of Maplets connected in Parallel, (c) A Reducelet followed by a series of Maplets, and (d) A set of Reducelts connected in parallel.

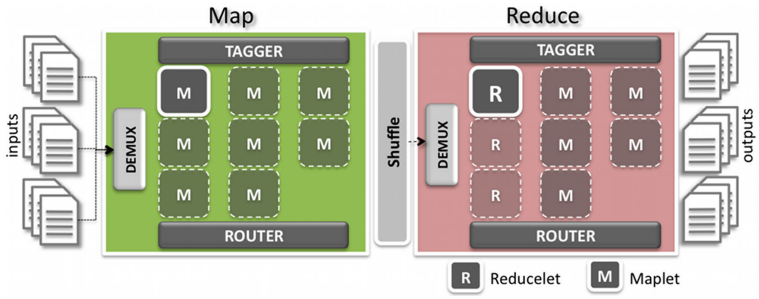

Figure 5.3: Internal components within a MR-LEGOS job that facilitates translating the job definition to the conventional MapReduce model.

The MR-LEGOS job model leverages the composition features mentioned above to provide a rich and refined model for the MapReduce job.

5.1.2 Translating MR-LEGOS Job Definition

This section describes how the refined job model proposed by MR-LEGOS is mapped to the conventional MapReduce model. We cover a set of main implementation aspects highlighting the procedure of such mapping.

Figure 5.3 shows the MR-LEGOS main Mapper and Reducer. The figure highlights the main components inside the Mapper and Reducer. These components enable the refined job definition provided by MR-LEGOS and are discussed below.

- **Inputs:** Inputs are grouped into logical units. A MR-LEGOS job can have one or more input units and the job definition describes how the input units are connected to

the Maplets within the job. The job definition also allows attaching separate readers to each input unit. The readers are responsible for splitting the input across multiple Mapper instances, understanding the input format and providing the steam of input key/value pairs to each Mapper instance.

- **Mapper DEMUX:** The MR-LEGOS Mapper has an input demultiplexing module. This module inspects every input key/value pair, figures out which logical input unit it belongs to and accordingly forwards the input key/value pair to the correct set of Maplets (i.e., the ones connected to this logical input unit).

- **Mapper ROUTER:** Each Maplet within the MR-LEGOS main Mapper has a routing module attached to its output. The router forwards the output key/value pair to every connected output Maplet. If the Maplet is connected to a Reducelet, then the output key/value pair is emitted as the intermediate key/value pair for the main Mapper. MR-LEGOS also provides a special type of routers called "conditional routers"; where the output key/value pairs can be selectively forwarded to the connected outputs. A set of Boolean functions ($f_i(key, value) \rightarrow true/false$), provided by the user, are associated with every connected output. The key/value pair is only forwarded to the output if the associated function is evaluated to true. The conditional routers offer a way for encoding a relatively more complex job semantics.

- **Mapper TAGGER:** Each Maplet within the main Mapper also has a tagging module which tags every output key/value pair. Each Maplet has a unique identifier (within the MR-LEGOS job) and this identifier is usually chosen as the tag identifier.

- **Reducer DEMUX:** The demultiplexing module in the main Reducer is responsible for receiving the stream of intermediate key/value pairs and splitting this stream across multiple Reducelets defined within the MR-LEGOS main Reducer. The demultiplexer uses the tags associated with the key/value pair to determine the appro-

priate recipient Reducelet. The design details for this demultiplexer are discussed in more details in Section 5.1.3.

- **Reducer ROUTER:** Every Reducelet and Maplet within the MR-LEGOS main Reducer has an associated routing module. This module is similar to the one mentioned above in the "Mapper ROUTER" discussion. Optionally, these routers can be chosen also to be conditional routers. If the Reducelet/Maplet is connected to an output unit then the output key/value pairs are emitted as final key/value outputs for the MR-LEGOS job. The output writer (see the following item) associated with this output unit is used.

- **Reducer TAGGER:** This is similar to the tagger discussed above for the main Mapper case.

- **Outputs:** Outputs are grouped into logical units. A MR-LEGOS job can have one or more output units and the job definition describes how they are connected to the Maplets/Reducelets within the job. The job definition also allows attaching separate writers to each output unit. These writers are responsible for writing the output key/value pairs in the appropriate format.

It is important to note that the discussions about the refined model for the Mapper and Reducer are also valid for the combiner.

5.1.3 Streaming intermediate values

In this section, we describe the design for splitting the stream of intermediate values received by the MR-LEGOS main Reducer into multiple streams for each Reducelet. In the MapReduce programming model, the Reduce function accepts an intermediate key and a set of values for that key. These intermediate values are supplied via an iterator to allow handling lists of values that are too large to fit in memory. The presented method extends

this design by providing a list of iterators, where values can be grouped into independent queues based on the tag. Multiple Reduce functions can independently iterate over these queues. The method is not memory bound and is advantageous when supporting multiple Reduce functions in a single MapReduce job.

We formulate the problem as follows: The Reducer receives an iterator over the set of values for a specific intermediate key. The requirement is to group these values based on the tag and stream these grouped values to multiple Reduce functions.

An obvious solution is for the Reducer to group values having the same tag into multiple lists (a list per unique tag), and provides an iterator over each list. The main problem with this solution is that it is memory bound. If the values doesn't fit in memory, it will be required to spill them to disk and then later read them back. This will present an I/O and performance degradation issue. The presented method addresses this issue and provides a simple and efficient solution, which is not memory bound and don not require writing/reading values to/from disk.

Blocking Queue The proposed method uses a blocking queue backed iterator. The blocking queue is a memory-based FIFO (first-in-first-out) data structure that mainly supports two methods: `add_object()`, for adding an object to the end of the queue and `remove_object()`, for retrieving and removing the object at the head of the queue. The queue has a predefined capacity `CMAX`. The `add_object()` method blocks if the queue is at its maximum capacity, while the `remove_object()` blocks if the queue is empty.

The architecture shown in Figure 5.4 depicts the details of how the intended functionality is achieved. Assume the MR-LEGOS job defines "n" Reducelets, where each Reducelet will have its own Reduce function; the architecture uses "n" execution threads ("receiver" threads); each will encompass a corresponding Reduce function and have an associated blocking queue iterator. Additionally, the architecture defines a single "sender" thread, which iterates through the received values one by one, and inspects the associ-

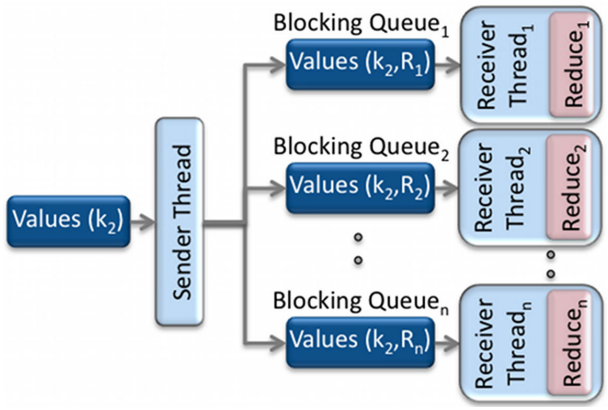

Figure 5.4: Detailed design of the Reducer DEMUX, where the stream of intermediate values is split into multiple streams, each corresponds to a Reducelet defined within the MR-LEGOS job.

ated tag and accordingly adds this value to the corresponding blocking queue using the `add_object()` operation. Each Reduce function is passed its corresponding blocking queue iterator to get access to its associated stream of values.

The architecture uses the tag to decide the forwarding criterion. In Figure 5.4, the notation $Values(k_2)$ refers to the whole stream of intermediate values associated with a specific intermediate key k_2, while the notation Values (k_2, R_n) refers to the stream split associated with Reducelet n. The Reduce function for an operation is called by the corresponding receiver thread and passed the blocking queue iterator, and hence can iterate through its grouped values. The concurrent and blocking nature of the design guarantees the synchronized functionality of Multiple Reduce functions.

The method is not memory bound since the additional used memory is a function of the number and maximum capacity of the used blocking queues. So the maximum queue size can be tuned to a value that avoids raising any memory or performance problems. The method also preserves the same iterator interface provided by the conventional MapReduce model, so it requires minimal modifications to migrate any legacy definitions to the MR-LEGOS design.

5.2 MR-LEGOS Data Elements

In this section, we will focus on discussing MR-LEGOS Data Elements (MR-LEGOS-DE), and how the MR-LEGOS refined model was used to build a data processing layer on top of MapReduce. MR-LEGOS-DE realizes the fact that there are sets of common logical components that usually exist within a single data processing MapReduce job and hence exposes an explicit model for structuring MapReduce jobs into standard reusable components that are meshed together to construct the job definition. The proposed approach maintains the ability of the user to write the conventional low-level MapReduce code, and at the same time, the refined structure provides a way of maintaining and reusing standard data processing components. The building units of the refined model remain the conventional Map and Reduce functions. However, they are grouped into logical components within the job definition. The presented approach offers a high-level, modular and easy way for defining complex and efficient MapReduce data processing jobs; it also allows the definition of multiple concurrent Mappers and Reducers within the same job, and this is advantageous from the perspective of disk access costs. In some scenarios it is required to perform multiple operations on the same data set. For example, having the relational table ($bonusID$, emp, $dept$, $amount$), it may be required to calculate the average bonus amount for each employee, and additionally, the total amount of bonuses for each department. Using the conventional MapReduce model, constructing two separate MapReduce jobs can perform these two operations, each reading the same data set. The use of multiple MapReduce jobs is apparently expensive, especially with large data sets. Using the proposed refined model, users can define multiple operations inside a single MapReduce job, and hence providing a convenient way of addressing such problems. The proposed framework makes use of the uniform interface of the Map and Reduce functions. This interface provides an intuitive way of connecting Mappers and Reducers. This is analogous to the knobs configuration in the LEGO blocks. MR-LEGOS leverages these features and provides a generic job design that is capable of defining complex micro-workflows inside a single MapReduce job.

5.2.1 MR-LEGOS-DE Job Definition

The objective of MR-LEGOS-DE is to provide a model suitable for defining data processing jobs within an ETL [81, 50] workflow. In order to come up with a convenient design, it is important to understand the context where the ETL job exists. An ETL workflow is constructed from a set of jobs; a job within the workflow may receive inputs from one or more preceding jobs, and may send its results to one or more following jobs. Within a single job, it is expected to find a central operation (e.g. joining two data-sets) in addition to a series of input or output specific transformations (e.g. filtering records). This logical organization is common to several state-of-the-art ETL tools (for example, see [39]).

The proposed approach defines a logical organization for the computations done within a MapReduce job; this organization enables the definition of standard processing components that can be easily plugged, reused and maintained. This feature can be viewed as defining a "micro" workflow inside the MapReduce job, where the user can mesh standard components or provide her own customized components. The user specifies the processing components and the connections between them and the framework takes care of data routing and computation coordination within these components. These processing components are standard Mappers and Reducers, which makes it easy to mesh components and reuse legacy code. This is analogous with how LEGO bricks can be used to build complex and large structures. Each individual brick is a very simple structure but because of the ability of connecting them in different configurations, it is easy to construct complex structures with various shapes. The MR-LEGOS refined design described in Section 5.1 offers an intuitive way to model such data processing problems.

5.3 Positioning MR-LEGOS

A data analyst is required to perform a data processing task on MapReduce. The task involve two data sets, shown in Figure 5.5, an $employee$ table listing the id, $name$, $deptId$

employees

id	name	deptId	salary
e1	Tom	d1	20
e2	John	d2	70
e3	Mary	d1	60
e4	Joe	d2	40

vehicles

eid	vmake
e1	Toyota
e2	Toyota
e3	Jaguar
e4	Lexus

Figure 5.5: An example employees table, showing the id, name, department Id and salary for each employee. An example vehicles tables, showing the employee Id and the vehicle make for each employee.

```
SELECT id, vMake
FROM employees JOIN vehicles ON id=eId
WHERE salary > 15;
```

```
SELECT deptID, AVG(salary)
FROM employees
GROUP BY depId
HAVING MIN(salary) > 10;
```

Figure 5.6: Two example SQL queries. A join query and an aggregation query using the tables shown in Figure 5.5.

and $salary$ for a company's employees, and a $vehicles$ table listing the employee id (eId) and the vehicle make ($vMake$) for the company's vehicles. It is required to evaluate the equivalent of the two SQL queries in Figure 5.6.

The data analyst has multiple alternatives:

Use a high-level language like Pig-Latin or Hive-SQL: Using these frameworks is highly convenient since the user describes the problem in a concise language and the framework takes care of translating the problem definition into Map and Reduce functions. In some cases, the user will get the queries executed and will be satisfied with the results and the performance, but what happens if the performance is not satisfactory or the results are not as expected. The user will start searching for the reasons and may ask questions like the following: What about using a different join strategy? May be the data is skewed, can I provide my own partitioner? Is the framework executing my queries as a single MapReduce job or multiple ones? How can I enforce using a single job?, etc. Such high-level languages generally don't provide this low-level control. Accordingly, one main issue will be the expressive power of such languages and how the constructs of the language are translated into MapReduce. If the user is facing the aforementioned problems, then a logical choice will be looking into other low-level alternatives (e.g. writing custom MapReduce code).

Write MapReduce code: By writing MapReduce code, the user is getting the maximum power and flexibility from the MapReduce programming model. The issue will be organizing, maintaining and reusing this code. The user may need to execute a slightly different query few days later, or an additional query will be added to the same job. An immediate solution is to write standard MapReduce jobs that perform well-defined operations. So assume the user starts to build a data processing library. For example, the user now has a job that performs hash join efficiently, another one for multi-way join or different aggregations; he also has simple jobs for filtering columns/records, etc. The user defined this library of jobs as standard Mappers and Reducers. However, for practical and performance reasons it may be needed to combine these operations inside one MapReduce job during execution. So a framework is needed that offer the capability of connecting these building blocks to construct more complex jobs. Here come the motivations of the proposed MR-LEGOS model.

Use MR-LEGOS: This alternative addresses the drawbacks associated with developing and maintaining custom MapReduce code. It should be noted that the proposed framework preserves the full MapReduce power and flexibility. For example, if the user defines an MR-LEGOS job that merely contains a single Maplet and Reducelet, then this is exactly equivalent to writing a single conventional MapReduce job. This is analogous to using LEGO, the system provides the brick and knob configurations and gives you several shapes of bricks, and the user may decide to build complex structures composed of hundreds of bricks or simple structures with a few bricks.

5.3.1 Simulation Example

To better understand the proposed MR-LEGOS model, we follow the example presented in the previous section and go through the processing steps. We want to construct a MR-LEGOS job that performs the equivalent of the aforementioned SQL queries. Figure 5.7 shows the combined query tree for the above mentioned SQL queries. The nodes in the

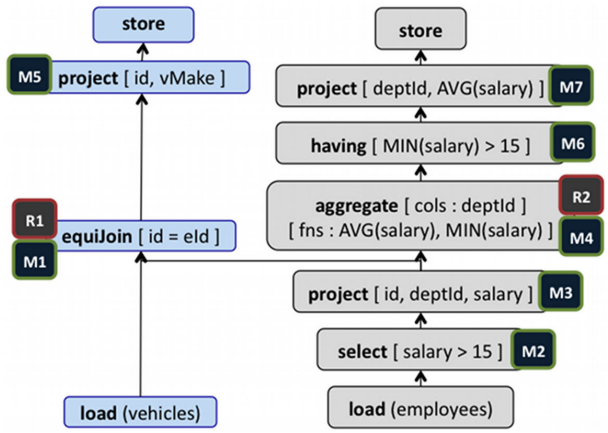

Figure 5.7: Combined query tree for the example SQL queries shown in Figure 5.6.

tree are tagged with the corresponding Maplets/Reducelets in Figure 5.8 to easily correlate the query semantics with MR-LEGOS job definition.

The diagram shown in Figure 5.8 depicts the definition for a MR-LEGOS job that concurrently performs these two queries. The diagram shows seven Maplets; $M1$ is connected to the $vehicles$ table, while $M2$ is connected to the $employees$ table. Maplet $M2$ is a $select$ Maplet to filter records (i.e., where $salary > 15$) followed by a $project$ Maplet for projecting only the required columns. Maplet $M1$ is a $join$ Maplet while $M4$ is an $aggregate$ Maplet. The output from the $project$ Maplet is connected to both the $join$ and $aggregate$ Maplets. The MR-LEGOS-DE library offers a toolbox of Maplets and Reducelets that can be used to specify different data processing tasks or even different techniques for the same operation (e.g. different join strategies). On the Reduce side, we have $join$ and $aggregate$ Reducelets coupled with their counter-parts on the Map side. The $join$ Reducelet, $R1$, is connected to a $project$ Maplet, $M3$, to get rid of the extra join key column, while the $aggregate$ Reducelet, $R2$, is connected to two Maplets connected in series, $M6$, implementing the SQL having semantics followed by a final $project$ Maplet, $M7$. The MR-LEGOS framework currently supports reading and writing XML and JSON specification of the job definition in addition to a relational model specification where Hibernate [70] is used to Map the job specification to its relational model.

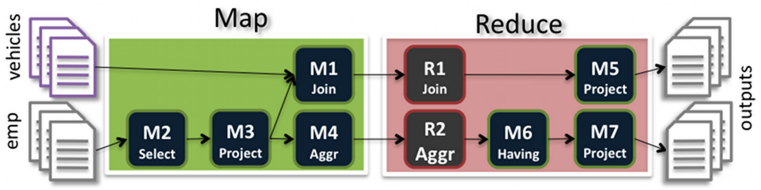

Figure 5.8: Single MR-LEGOS job definition for the two SQL queries shown in Figure 5.6.

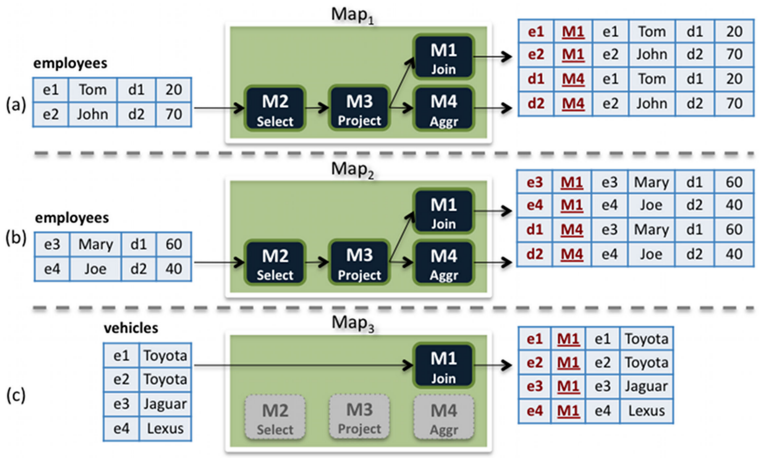

Figure 5.9: Map-side of the simulation, where the employees table was divided into two splits and two Map tasks are processing these splits, while the vehicles table is processed by the third Map task.

With the aforementioned MR-LEGOS job definition, let us go through the simulation run of the job. Figure 5.9 depicts the Map-side execution details. As the Map operation is parallelized, the input file set is first split to several pieces called "file splits". If an individual file is so large that it will affect seek time it will be split to several splits [64]. For the sake of the running example, we assume that the MapReduce framework decided to create three file splits and hence three Mapper instances. The $employee$ table was split into two file splits, while the $vehicles$ table remained as a single file split; the contents of these splits are shown in the left-hand side of Figure 5.9.

The Mapper DEMUX (described in Section 5.1.2) identifies the file split the Mapper instance is reading from, and hence identifies the associated logical input table. Mapper instances $Mapper_1$ and $Mapper_2$ are reading file splits associated with the $employee$ table and hence all records are forwarded to the $M2$ Maplet, while Mapper instance $Mapper_3$

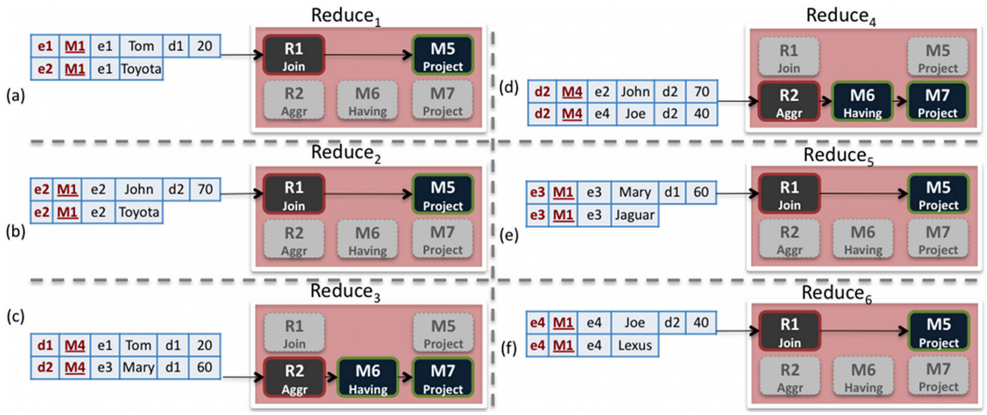

Figure 5.10: Reduce-side of the simulation, where six Reduce tasks are processing the grouped intermediate values produced by the Map tasks shown in Figure 5.9.

is reading a $vehicles$ table file split and accordingly all records are passed to $M1$. This is the reason $M2$, $M3$ and $M4$ are shown as inactive in $Mapper_3$. The figure identifies inactive components by the light gray color. Following the details described in Section 5.1.2, the $employees$ table's records are tagged and then forwarded to both the $join$ and the $aggregate$ Maplets and Reducelets. The records of the $vehicles$ table are tagged by $M1$ tag and only forwarded to the $join$ Reducelet. The aggregate Maplet is shown inactive in $M3$. The $join$ and $aggregate$ Maplets, respectively, publish the join and aggregate columns for each received record as the intermediate key and the whole record as the intermediate value. The intermediate outputs are shown in the right-hand side of Figure 5.9; the intermediate key and the tag are respectively shown to the left of each record.

Figure 5.10 depicts the Reduce-side execution details. The MapReduce library takes care of partitioning intermediate outputs and grouping all values having the same intermediate key. The grouped values are shown to the left side of Figure 5.10. Note that, values for different intermediate keys can end-up in the same Reducer (based on the partitioner in the Map-side), and in such case the Reduce function is called as many times as there are distinct intermediate keys. The Reduce instances ($Reducer_1$..$Reducer_6$) shown in the figure represent distinct calls to the Reduce function (either in the same Reducer or separate Reducers). The Reducer DEMUX (described in Section 5.1.2) inspects the tag for

each received record and forwards the record to the correct Reducelet, where the Reduce-side of the operation is executed. In the $join$ Reducelet, the values are grouped based on the input unit tag and then a cross product between the two groups is evaluated, while in the $aggregate$ Reducelet, the aggregate functions (AVG and MIN) are calculated for the group. It can be seen that both operations are concurrently running on different Reducers. Finally, the output from each Reducelet is forwarded to the connected Maplets. The framework uses multiple output collectors to realize the different logical output units. So in our example, we have two output collectors, one for each logical output unit. The Reducer ROUTER takes care of forwarding the output to the corresponding output collector. The framework takes care of generating as many output collectors as the number of defined logical output units inside the MR-LEGOS job.

In the above example, we assumed that the intermediate keys for each operation don't overlap and hence for each Reduce instance we had only one active Reducelet. If the intermediate keys overlap, we may have multiple active Reducelets in the same main Reducer instance. For example, assume that the two keys $e1$ and $d1$ were equal, and accordingly the values for them were grouped as a single stream of intermediate values for the main Reduce instance (i.e. $Reducer_1$ and $Reducer_3$ were merged into a single Reduce instance). In this case, it is required to devise a mechanism to resolve this issue since it is required to have a separate stream on intermediate values for each defined Reducelet. The partitioner in the Map-side can be designed to control this issue by preventing values having equal keys and equal Reducelets tags to be inserted into the same partition (this requires enforcing the number of Reducers to be at least equal to the number of defined operations inside the job). A more general solution is to split the single stream of intermediate values into two steams (one for each Reducelet), this splitting mechanism was described earlier in Section 5.1.3.

5.4 Experimental Results

We present a number of experiments conducted to assess the performance and scalability of the proposed model. Section 5.4.1 discusses the experiments carried on a single compute node, while the results for multi-node cluster are discussed in Section 5.4.2.

5.4.1 Single-Node Cluster

The first set of experiments were carried on a single compute node, the objective was to investigate the basic characteristics of the model, in isolation from communication overheads that exist in a multi-node cluster. The experiments were carried on a MAC OS X 10.5, with 2.2 GHz Intel Core 2 Duo processor, 4GB RAM and 120GB HD, running Hadoop 0.20.1. The steps for setting up Hadoop MapReduce on a single-node can be found at [125].

As previously mentioned, the MR-LEGOS model degenerates to the conventional MapReduce model, if the user defines a MR-LEGOS job that is merely composed of a single Maplet and Reducelet. Since MR-LEGOS is designed as a generic model with the presence of the standard components (e.g. demultiplexing, routing and tagging modules), additional cost is expected when using MR-LEGOS in such degenerate case. The first experiment in this section assesses this added overhead. We used a set of data processing examples with data of varying sizes and measured the execution time when the Maplet and Reducelet were part of a MR-LEGOS job versus the case of using these Maplet and Reducelet as the main Mapper and Reducer in a conventional MapReduce job. Figure 5.11 shows the results, the x-axis shows the input data sizes as multiples of the initial size (i.e. at point 1), while the execution time is shown on the y-axis. The results show that the execution time for MR-LEGOS was on average $2 - 3\%$ more than the execution time for the same job using the conventional model. The input data sizes in this experiment varied from 244KB to 3,416KB, a single Mapper and Reducer instance were used. Such overhead was generally acceptable, given the advantages of using MR-LEGOS.

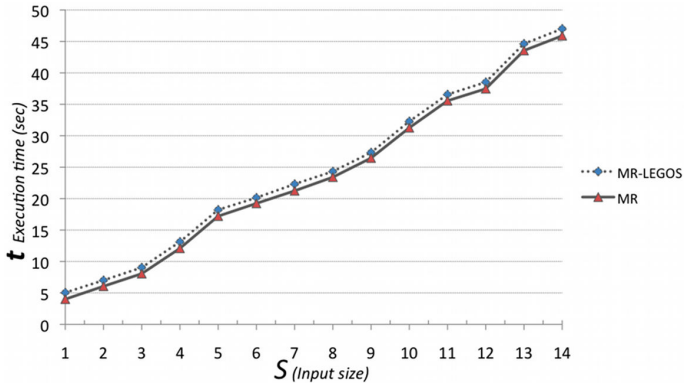

Figure 5.11: Execution time as a function of input data sizes for both the MR-LEGOS and the conventional MR cases in the single-node cluster case.

To investigate the composition scalability of the proposed model, we evaluated the behavior of the proposed MR-LEGOS model as a function of varying input data sizes and variable number of Reducelets packed within the MR-LEGOS job. The charts in Figure 5.12 illustrate the results for this experiment. The number of concurrently running Reducelets (i.e. the number of threads within a main Reduce task) is shown on the x-axis. The y-axis represents the relative execution time $\tau_n = \frac{t_n}{n \times t_1}$, where t_n is the execution time for an n-Reducelets job, and t_1 is the execution time for a single Reducelet job for the same data size. The charts ($1x$ .. $5x$) represent results for varying input data sizes (the input data size in $2x$ is double the one in $1x$, etc.). The input data sizes varied from 85KB to 425KB. Inspecting the results for $1x$, it can be seen that τ decreases as the number of threads slightly increase but then increases as the number of threads further increase. This increase is associated with the fact that at some point the increased number of threads becomes an overhead for their main Reduce task process. This effect becomes significant as the data sizes increase (e.g. the chart for the $5x$ case). To practically avoid this limitation, the Map partitioner can be designed to limit the number of concurrently running Reducelets within a single Reduce instance.

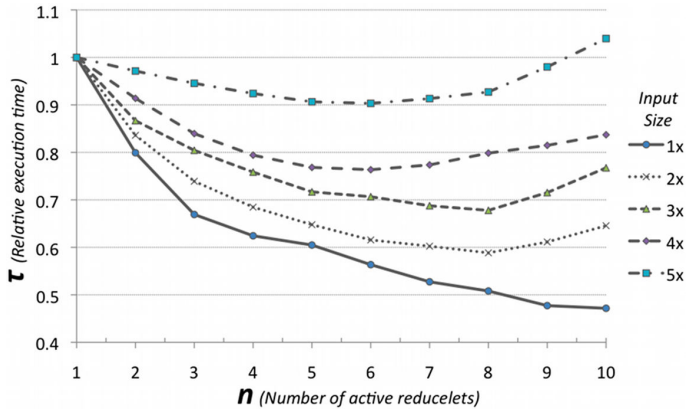

Figure 5.12: Relative execution time as a function of no. Reducelets within a Reduce task for different data sizes in the single-node cluster case.

5.4.2 Multi-Node Cluster

In the second set of experiments, we investigated the characteristics of MR-LEGOS model on a shared Multi-Node Hadoop 0.20.1 cluster. The cluster has 4000 nodes, each node has a 2x Quad Core 1.86GHz Intel Xeon processor, 16GB RAM, and 4x 750GB HD. For detailed steps on setting up a Multi-Node Hadoop cluster, the interested reader can refer to [98].

Multiple users/jobs can concurrently use a shared Hadoop cluster. Accordingly, jobs can be queued for a period of time before execution, the queuing time depends on the job priority and the load on the cluster (i.e., the total number of running tasks in the cluster). Also, the job runtime varies based on the cluster load, and based on the location where the Map/Reduce tasks ran relative to their input data locations. Hadoop generally tries to run the tasks as close to their input data as possible, however, there is no guarantees, as locations can vary based on nodes availability. To filter such noise, all queueing times were eliminated from the reported results. Also, we ran each job for at least three times (at different times of the day) and used the median results (to eliminate noisy outliers).

The first experiment investigated the runtime of MR-LEGOS jobs compared to their equivalent MR jobs to assess any runtime overheads on a Multi-Node cluster. The results are shown in Figure 5.13, the x-axis shows the input data sizes as multiples of the initial

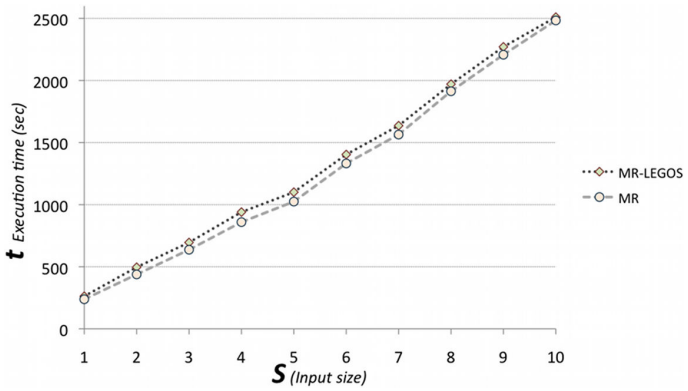

Figure 5.13: Execution time as a function of input data sizes for both the MR-LEGOS and the conventional MR cases in the multi-node cluster case.

data size, while the y-axis shows the execution time (in seconds). The input data sizes varied from 60MB (at point 1) to 600MB (at point 10). For each input data size, both the MR-LEGOS job and its equivalent MR job were started at the same time, to have them share a common cluster status during execution. For all jobs in this experiment, we used 10 Map tasks and 50 Reduce tasks. It can be noticed that as the size of the input data increased, the MR-LEGOS overhead becomes negligible relative to the job execution time. We have repeated the experiments for different data sizes and job configurations, the overhead varied between 1% and 3% on average. The communication overheads in the multi-node cluster were common for both the MR-LEGOS and conventional MR cases, hence the results of this multi-node experiment were aligned with the single-node case (see Figure 5.11). However, the multi-node experiment gave more insights on the fact that the overhead becomes negligible for relatively larger jobs.

One of the key elements of the MR-LEGOS model is the ability of composing Maplets and Reducelets within a single Map or Reduce task. In the second experiment, the objective was to explore the limits in terms of scalability in composing these elements within a single task. In contrast with the experiment in Section 5.4.1, the focus here was to investigate the multi-node case to see if the communication overheads may expose other characteristics. We also focused on compositions that result in multi-threaded jobs as opposed to single-threaded compositions (i.e., only connecting elements in series), since

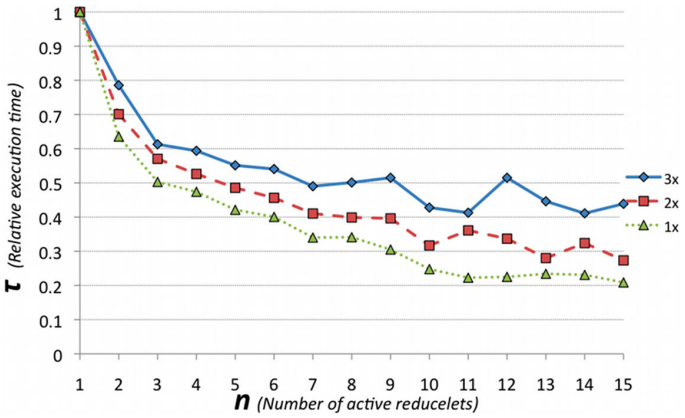

Figure 5.14: Relative execution time as a function of no. Reducelets within a Reduce task for different data sizes in the multi-node cluster case.

multi-threaded compositions are more expensive. The results shown in Figure 5.14 used an input data of varying sizes 60, 120 and 180MB corresponding to the charts labeled 1x, 2x and 3x, respectively. The intermediate data sizes ranged from 60MB to 2.7GB, each job used 10 Map tasks and 50 Reduce tasks. The x-axis shows the number of Reducelets within a single Reduce task, we varied this number from 1 to 15 Reducelets. This range was practically sufficient, as the maximum number of Reducelets within a Reduce task was always less than 5 in all production scenarios we have seen. The y-axis shows the relative execution time τ (first introduced in Section 5.4.1). For a job with n Reducelets, the relative execution time is the job execution time divided by the sum of execution times for n equivalent jobs (each having a single Reducelet). As the number of Reducelets increased, the execution time also increased, while the relative execution time (τ) decreased, however, the rate of such decrease dropped as the number of Reducelets increased. Ideally we want τ to be smaller than 1 as this means that packing the job with multiple Reducelets had a better runtime than running each Reducelet in a separate job. Compared to the results observed in the single-node experiment (Figure 5.12), the multi-node experiment always exhibited a τ value smaller than 1 for the whole range of Reducelets. This behavior can be attributed to the fact that the node configuration in the multi-node cluster, in terms of processing power, main memory, etc., comfortably handled the increased number of threads without

exhibiting any negative overheads as observed in the single-node case (See the 5x-series in Figure 5.12).

The drop in the τ decrease rate, as the number of Reducelets increased, in both the single-node and multi-node cases, can be also attributed to Amdahl's Law [14, 71], this law is concerned with the speedup achievable from an improvement to a computation that affects a proportion p of that computation where the improvement has a speedup of n. Amdahl's law states that the overall speedup of applying the improvement will be:

$$Speedup(p, n) = \frac{1}{(1 - p) + \frac{p}{n}} \tag{5.1}$$

In our case, p represents the actual data processing done inside the Reducelet, while $1-p$ is the extra sequential overhead done in the encompassing Reducer (i.e., multiplexing, routing and tagging operations), in addition to the multi-threading overhead. The results in the multi-node case also show an increased rate of fluctuations in the jobs' runtime (especially when #Reducelets > 10), this behavior can be attributed to both the communication overheads in the cluster and the increased number of threads.

Chapter 6

Conclusions and Future Directions

6.1 Conclusions

In this thesis, we studied the problems of data management, integration and processing in grid/cloud environments, with special emphasis on bioinformatics applications. The broader context of designing a services oriented architecture (SOA) for data management and integration was studied, identifying the main components/services for realizing this architecture.

Researchers in bioinformatics were excited about the successful mapping of the human genome. These large amounts of genetic data represents a powerful tool that could be used to put an eventual end to many diseases. But integrating the data so that researchers could access and analyze it quickly had always been a challenge. The BioFederator is a web services-based data federation architecture for bioinformatics applications. The system helps researchers manage and integrate data efficiently in a distributed/cloud computing environment and makes it easy to gain quick access to the critical data needed for analytical purposes. Based on collaborations with bioinformatics researchers, several domain-specific data federation challenges and needs were identified. The BioFederator addresses such challenges and provides an architecture that incorporates a series of utility services. These

address issues like automatic workflow composition, domain semantics, and the distributed nature of the data. It also incorporates a series of data-oriented services that facilitate the actual integration of data. The proposed design, services, and usage scenarios were discussed in detail [115, 19].

We paid special attention to a set of important applications in the bioinformatics domain. For the first time, we presented a whole genome prediction of nucleosome exclusion regions for the human genome. The resulting annotation was studied and correlated with tissue specificity, gene density and other important gene regulation features [80, 116]. Also the output results were made available to the scientific community as part of the University of California at Santa Cruz (UCSC) Genome Browser custom data tracks. The UCSC Genome Browser is internationally recognized as one of the most important bioinformatics data repositories.

Within the BioFederator broader context, we studied also the problem of combining several heterogeneous data representations into a unified non-redundant representation, this is known as the schema integration problem. Schema integration is the problem of creating a unified target schema based on a set of existing source schemas and based on a set of correspondences that are the result of matching the source schemas. Previous methods for schema integration rely on the exploration, implicit or explicit, of the multiple design choices that are possible for the integrated schema. Such exploration relies heavily on user interaction; thus, it is time consuming and labor intensive. Furthermore, previous methods have ignored the additional information that typically results from the schema matching process, that is, the weights and in some cases the directions that are associated with the correspondences.

In this thesis, we proposed a more automatic approach to schema integration that is based on the use of directed and weighted correspondences between the concepts that appear in the source schemas. We designed a novel similarity measure used to quantify the distance between schema concepts in the schema integration problem. This similarity mea-

sure has significant importance in metadata management in distributed computing systems as it facilitates the process of federating data from multiple autonomous data sources and generating a unified non-redundant representation of the data. The new measure offers an enhanced set of integration decisions as subsumption in addition to merge decisions can be discovered. A key component of our approach is a top-k ranking algorithm for the automatic generation of the best candidate schemas. The algorithm gives more weight to schemas that combine the concepts with higher similarity or coverage. Thus, the algorithm makes certain decisions that otherwise would likely be taken by a human expert. The algorithm constitutes an important advancement compared to previous algorithms as it runs in polynomial time and has good performance in practice. The proposed methods and algorithms were compared to the state of the art approaches, the comparisons relied on complexity analysis and performance evaluation [114].

We also studies data processing models on grid environment, we highlighted several issues in the existing MapReduce abstraction approaches and alternatively proposed a novel refined MapReduce model that addresses the maintainability and reusability issues, without sacrificing the low-level controllability offered by directly writing MapReduce code. We presented the details and our experiences in designing and building MR-LEGOS; an explicit model for composing MapReduce constructs from simpler components, namely, "Maplets", "Reducelets" and optionally "Combinelets". MR-LEGOS offers an easy way of re-using and maintaining standard data processing components, in addition to the ease of plugging custom components. The refined model can be easily tested, validated and debugged, since it is composed of standard and relatively simpler components. Compared to other high-level abstractions, MR-LEGOS offers a powerful abstraction layer. These high-level tools translate user queries into MapReduce jobs, but the user doesn't have low-level control over the details of these constructed jobs. Additionally, each of these high-level tools is limited by the expressive power of its query language and how these queries are mapped to MapReduce jobs. We showed that the refined model is suitable for defining

complex ETL grid jobs, and illustrated the design details through example simulations and performance experiments [117].

6.2 Future Directions

One important future direction is to apply and extend schema integration techniques such as the ones developed in this thesis to situations where the number of schemas to be integrated is much larger (e.g., in the hundreds) and where schemas rapidly evolve and change. In particular, we would like to be able to automate the problem of generating unified schemas when extracting and integrating large data sets from the web (e.g., from DBPedia, Freebase, etc). Even when the data has structure, it is often the case that the schema underlying an entity of interest varies, sometimes significantly, from one individual object to another, and coming up with a unified schema will pose challenges.

From an applications perspective, an important future direction is applying the proposed data integration methods and algorithms to investigate other bioinformatics problems. For example, pharmacogenomics is a branch of bioinformatics dealing with the influence of genetic variation on drug response in patients. Approaches investigating such influences promise the advent of "personalized medicine", in which drugs and drug combinations are optimized for each individual's unique genetic makeup. To make "personalized medicine" decisions, information from multiple heterogeneous data sources needs to be incorporated; for example OMIM, dbSNP and dbGaP from NCBI, Haplotype data from the HapMap project, Human Gene Mutation and TRANSFAC databases from BioBase, in addition to PHARMKGB.

In this thesis, we focused on using MR-LEGOS in grid data processing applications, so one important future direction is to apply the MR-LEGOS model in other application domains (e.g. machine learning). Also, investigating the possible extension of MR-LEGOS micro-workflow specifications to include advanced features like loop-backs can be an inter-

esting future direction. Integrating MR-LEGOS with the other high-level abstraction layers can be also important.

Bibliography

[1] An Exchange Format for Feature Description (GFF). http://www.sanger.ac.uk/Software/formats/GFF/.

[2] BioJava. http://biojava.org/.

[3] SymAtlas. http://symatlas.gnf.org/SymAtlas/.

[4] UCSC genome browser custom tracks page. http://genome.ucsc.edu/goldenPath/customTracks/custTracks.html.

[5] Wiggle Track Format (WIG). http://genome.ucsc.edu/goldenPath/help/wiggle.html.

[6] Biohavasu. http://phanxipan.eas.asu.edu, 2003.

[7] The encode project consortium. the encode (encyclopedia of dna elements) project. *Science*, 306:636–640, 2004.

[8] The encode project consortium. identification and analysis of functional elements in 1% of the human genome by the encode pilot project. *Nature*, 447:799–816, 2007.

[9] D. J. Abadi. Data management in the cloud: Limitations and opportunities. *IEEE Data Eng. Bull.*, 32(1):3–12, 2009.

[10] A. Abouzeid, K. Bajda-Pawlikowski, D. J. Abadi, A. Rasin, and A. Silberschatz. Hadoopdb: An architectural hybrid of mapreduce and dbms technologies for analytical workloads. *PVLDB*, 2(1):922–933, 2009.

[11] G. Aloisio, M. Cafaro, S. Fiore, and M. Mirto. The grid-dbms: Towards dynamic data management in grid environments. *Information Technology: Coding and Computing, International Conference on*, 2:199–204, 2005.

[12] AmazonEC2. Amazon Elastic Compute Cloud (Amazon EC2). http://aws.amazon.com/ec2/.

[13] AmazonS3. Amazon Simple Storage Service (Amazon S3). http://aws.amazon.com/s3/.

[14] G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, pages 483–485, 1967.

[15] M. Angermayr and W. Bandlow. Permanent nucleosome exclusion from the Gal4p-inducible yeast GCY1 promoter. *The Journal of biological chemistry*, 278:11026–11031, 2003.

[16] M. Angermayr, U. Oechsner, and W. Bandlow. Reb1p-dependent DNA Bending Effects Nucleosome Positioning and Constitutive Transcription at the Yeast Profilin Promoter. *The Journal of biological chemistry*, 278:17918–17926., 2003.

[17] G. Antoniu. Autonomic cloud storage: Challenges at stake. *CISIS*, page 481, 2010.

[18] Apache. Apache Software Foundation. `http://hadoop.apache.org/`.

[19] R. Badia, G. Dasgupta, O. Ezenwoye, L. Fong, H. Ho, S. Khuri, Y. Liu, S. Luis, A. Praino, J.-P. Prost, A. Radwan, S. M. Sadjadi, S. Shivaji, B. Viswanathan, P. Welsh, and A. Younis. Innovative Grid Technologies Applied to Bioinformatics and Hurricane Mitigation. In *High Performance Computing and Grids in Action*, pages 436–462. IOS Press, 2008.

[20] P. Baker, A. Brass, S. Bechhofer, C. Goble, N. Paton, and R. Stevens. TAMBIS: Transparent Access to Multiple Bioinformatics Information Sources. In *Sixth International Conference on Intelligent Systems for Molecular Biology*, 1998.

[21] T. Barrett and et. al. NCBI GEO: mining tens of millions of expression profiles database and tools update. *Nucleic Acids Research*, 35:D760–D765, 2006.

[22] C. Batini, M. Lenzerini, and S. B. Navathe. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Comput. Surv.*, 18(4):323–364, 1986.

[23] S. Bergamaschi, S. Castano, and M. Vincini. Semantic integration of semistructured and structured data sources. *ACM SIGMOD Record*, 28(1), 1999.

[24] S. Boulakia, O. Biton, S. Cohen, Z. Ives, V. Tannen, and S. Davidson. SHARQ Guide: Finding relevant biological data and queries in a peer data management system. 2006.

[25] P. Brown, P. J. Haas, J. Myllymaki, H. Pirahesh, B. Reinwald, and Y. Sismanis. Toward Automated Large-Scale Information Integration and Discovery. In *Data Management in a Connected World*, pages 161–180, 2005.

[26] P. Buneman, S. B. Davidson, and A. Kosky. Theoretical Aspects of Schema Merging. In *EDBT*, pages 152–167, 1992.

[27] D. Buttler, M. Coleman1, T. Critchlow1, R. Fileto, W. Han, L. Liu, C. Pu, D. Rocco, and L. Xiong. Querying Multiple Bioinformatics Data Sources: Can Semantic Web Research Help? *ACM SIGMOD Record*, 31(4), 2002.

[28] A. Cali, D. Calvanese, G. D. Giacomo, and M. Lenzerini. On the Expressive Power of Data Integration Systems. 2002.

[29] H. Cao, H. R. Widlund, T. Simonsson, and M. Kubista. TGGA Repeats Impair Nucleosome Formation. *J. Mol. Biol.*, 281:253–120, 1998.

[30] P. Carninci, A. Sandelin, B. Lenhard, S. Katayama, K. Shimokawa, J. Ponjavic, C. A. M. Semple, M. S. Taylor, P. G. Engstrm, M. Frith, A. R. R. Forrest, W. B. Alkema, S. L. Tan, C. Plessy, R. Kodzius, T. Ravasi, T. Kasukawa, S. Fukuda, M. K. Katayama, Y. Kitazume, H. Kawaji, C. Kai, M. Nakamura, H. Konno, K. Nakano, S. M. Tabar, P. Arner, A. Chesi, S. Gustincich, F. Persichetti, H. Suzuki, S. M. Grimmond, C. A. Wells, V. Orlando, C. Wahlestedt, E. T. Liu, M. Harbers, J. Kawai, V. B. Bajic, D. A. Hume, and Y. Hayashizaki. Genome-wide analysis of mammalian promoter architecture and evolution. *Nat Genet*, 38:626–635, 2006.

[31] Cascading. An API for defining data processing workflows on Hadoop. `http://www.cascading.org/`.

[32] S. Castano, V. D. Antonellis, and S. D. C. diVemercati. Global viewing of heterogeneous data sources. *IEEE Trans. Data Knowl. Eng.*, 13(2):277–297, 2001.

[33] L. Chiticariu, M. A. Hernández, L. Popa, and P. G. Kolaitis. Semi-Automatic Schema Integration in Clio. In *VLDB*, pages 1326–1329, 2007.

[34] L. Chiticariu, P. G. Kolaitis, and L. Popa. Interactive Generation of Integrated Schemas. *Submitted for publication*, 2007.

[35] Cloud. Cloud computing definition, national insitute of standards and technology, version 15. `http://csrc.nist.gov/groups/SNS/cloud-computing/index.html`.

[36] T. M. Connolly and C. E. Begg. *Database Systems: A Practical Approach to Design, Implementation and Management*. Addison-Wesley, 2004.

[37] B. F. Cooper, E. Baldeschwieler, R. Fonseca, J. J. Kistler, P. P. S. Narayan, C. Neerdaels, T. Negrin, R. Ramakrishnan, A. Silberstein, U. Srivastava, and R. Stata. Building a cloud for yahoo! *IEEE Data Eng. Bull.*, 32(1):36–43, 2009.

[38] S. J. Cooper, N. D. Trinklein, E. D. Anton, L. Nguyen, and R. M. Myers. Comprehensive analysis of transcriptional promoter structure and function in 1% of the human genome. *Genome Res.*, 16(1):1–10, 2006.

[39] Datastage. IBM Datastage. `http://www-01.ibm.com/software/data/infosphere/datastage/`.

[40] S. Davidson, J. Crabtree, B. Brunk, J. Schug, V. Tannen, C. Overton, and C. Stoeckert. K2/Kleisli and GUS: Experiments in Integrated Access to Genomic Data Sources. *IBM Systems Journal*, 40(2):512–531, 2001.

[41] S. Davidson, C. Overton, and P. Buneman. Challenges in Integrating Biological Data Sources. *Journal of Computational Biology*, 2(4), 1995.

[42] S. Davidson and L. Wong. The Kleisli Approach to Data Transformation and Integration. 2001.

[43] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *OSDI*, pages 137–150, 2004.

[44] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.

[45] S. Dessloch, M. A. Hernández, R. Wisnesky, A. Radwan, and J. Zhou. Orchid: Integrating schema mapping and etl. In *ICDE*, pages 1307–1316, 2008.

[46] A. Doan, P. Domingos, and A. Levy. Learning source descriptions for data integration. pages 81–92, 2000.

[47] A. Doan, J. Madhavan, P. Domingos, and A. Halevy. Learning to map between ontologies on the semantic web. pages 662–673, 2002.

[48] M.-P. Dubuisson and A. K. Jain. A Modified Hausdorff Distance for Object Matching. In *Proc. Int. Conf. on Pattern Recognition*, pages 566–568, 1994.

[49] O. M. Duschka and M. R. Genesereth. Answering recursive queries using views. In *PODS*, pages 109–116, 1997.

[50] ETL. Extract, Transform and Load (ETL). `http://en.wikipedia.org/wiki/Extract,\_transform,\_load`.

[51] A. G. Fernandeza and J. N. Anderson. Nucleosome positioning determinants. *J Mol Biol.*, 371(3):649–668, 2007.

[52] D. Florescu, A. Levy, and A. Mendelzon. Database Techniques for the World-Wide Web: A Survey. *ACM SIGMOD Record*, 27(3):59–74, 1998.

[53] M. Friedman, A. Levy, and T. Millstein. Navigational Plans For Data Integration. pages 67–73, 1999.

[54] M. Friedman, A. Levy, and T. Millstein. Navigational plans for data integration. In *16th National Conference on Artificial Intelligence (AAAI)*, 1999.

[55] A. Gal. Managing Uncertainty in Schema Matching with Top-K Schema Mappings. *J. Data Semantics*, 6:90–114, 2006.

[56] A. Gal, A. Anaby-Tavor, A. Trombetta, and D. Montesi. A framework for modeling and evaluating automatic semantic reconciliation. *VLDB Journal*, 14(1):50–67, 2005.

[57] M. Ganapathi, P. Srivastava, S. K. D. Sutar, K. Kumar, D. Dasgupta, G. P. Singh, V. Brahmachari1, and S. K. Brahmachari. Comparative analysis of chromatin landscape in regulatory regions of human housekeeping and tissue specific genes. *Crit Rev Eukaryot Gene Expr*, 6:126, 2005.

[58] H. Garcia-Molina and et. al. The TSIMMIS Approach to Mediation: Data Models and Languages. *Journal of Intelligent Information Systems*, 8(2):117–132, 1997.

[59] J. A. Greenbaum, S. C. Parker, and T. D. Tullius. Detection of DNA structural motifs in functional genomic elements. *Genome Research*, 17:940–946, 2007.

[60] F. E. Grubbs. Procedures for detecting outlying observations in samples. *Technometrics*, 11:1–21, 1969.

[61] J. Gutierrez, R. Paredes, F. Cruzat, D. A. Hill, A. J. van Wijnen, J. B. Lian, G. S. Stein, J. L. Stein, A. N. Imbalzano, and M. Montecino. Chromatin Remodeling by SWI/SNF Results in Nucleosome Mobilization to Preferential Positions in the Rat Osteocalcin Gene Promoter. *THE JOURNAL OF BIOLOGICAL CHEMISTRY*, 282(18):9445–9457, 2007.

[62] L. Haas, P. Schwarz, P. Kodali, E. Kotlar, J. Rice, and W. Swope. DiscoveryLink: A System for Integrated Access to Life Sciences Data Sources. *IBM Systems Journal*, 40(2):489–511, 2001.

[63] L. M. Haas, M. A. Hernández, H. Ho, L. Popa, and M. Roth. Clio Grows Up: From Research Prototype to Industrial Tool. In *SIGMOD*, pages 805–810, 2005.

[64] Hadoop-MapReduce. Hadoop MapReduce Tutorial. `http://hadoop.apache.org/core/docs/r0.20.0/mapred\_tutorial.html`.

[65] A. Halevy. Answering Queries Using Views: A Survey. *VLDB Journal*, pages 270–294, 2001.

[66] A. Y. Halevy, Z. G. Ives, J. Madhavan, P. Mork, D. Suciu, and I. Tatarinov. The Piazza Peer Data Management System. *IEEE Trans. Knowl. Data Eng.*, 16(7):787–798, 2004.

[67] H. Hamacher and M. Queyranne. K-best solutions to combinatorial optimization problems. *Annals of Operations Research*, 4:123–143, 1985/6.

[68] J. Hammer and M. Schneider. Genomics Algebra: A New, Integrating Data Model, Language, and Tool Processing and Querying Genomic Information. 2003.

[69] T. Hern and S. Kambhampati. Integration of biological sources: Current systems and challenges ahead. *Sigmod Record*, 33:51–60, 2004.

[70] Hibernate. Relational Persistence for Java. `https://www.hibernate.org/`.

[71] M. D. Hill and M. R. Marty. Amdahls law in the multicore era. *IEEE COMPUTER*, 2008.

[72] D. Hull and et. al. Taverna, A tool for building and running workflows of services. *Nucleic Acids Research*, 34:w729–w732, 2006.

[73] D. P. Huttenlocher, G. A. Klanderman, and W. J. Rucklidge. Comparing Images Using the Hausdorff Distance. *IEEE Trans. PAMI*, 15:850–863, 1993.

[74] Jaql. Query Language for JavaScript(r) Object Notation (JSON). `http://www.jaql.org/`.

[75] H. Jiang, H. Ho, L. Popa, and W.-S. Han. Mapping-Driven XML Transformation. In *16th International World Wide Web Conference*, 2007.

[76] K. Kadota, S.-I. Nishimura, H. Bono, S. Nakamura, Y. Hayashizaki, Y. Okazaki, and K. Takahashi. Detection of genes with tissue-specific expression patterns using Akaike's information criterion procedure. *Physiol Genomics*, 12:251–259, 2003.

[77] K. Kadota, J. Ye, Y. Nakai, T. Terada, and K. Shimizu. ROKU: a novel method for identification of tissue-specific genes. *BMC Bioinformatics*, 7:294, 2006.

[78] O. Kel-Margoulis and et. al. *Information Processing and Living Systems:Databases on Gene Regulation*. Imperial College Press, London, 2005.

[79] W. Kent, C. Sugnet, T. Furey, K. Roskin, T. Pringle, A. Zahler, and D. Haussler. The Human Genome Browser at UCSC. *Genome Research*, 12(6):996–1006, 2002.

[80] S. Khuri, A. Radwan, P. Luykx, and A. Younis. Nucleosome exclusion regions across the human genome. *American Society of Human Genetics (ASHG) 57th Annual Meeting, San Diego, California, 23-27 October*, 2007.

[81] R. Kimball and J. Caserta. *The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleanin*. John Wiley & Sons, 2004.

[82] T. Klein and et. al. Integrating Genotype and Phenotype Information: An Overview of the PharmGKB Project. *The Pharmacogenomics Journal*, 1:167–170, 2001.

[83] Z. Lacroix, F. Naumann, L. Raschid, and M.-E. Vidal. Exploring Life Sciences Data Sources. 2003.

[84] J. Larson, S. Navathe, and R. ElMasri. A theory of attribute equivalence in databases with application to schema integration. *IEEE Trans. Software Eng.*, 16(4):449–463, 1989.

[85] W. Lee, D. Tillo, N. Bray, R. H. Morse, R. W. Davis, T. R. Hughes, and C. Nislow. A high-resolution atlas of nucleosome occupancy in yeast. *Nature Genetics*, 39:1235–1244, 2007.

[86] M. Lenzerini. Data Integration: A Theoretical Perspective. *ACM Symposium on Principles of Database Systems (PODS)*, 2002.

[87] V. G. Levitsky, O. A. Podkolodnaya, N. A. Kolchanov, and N. L. Podkolodny. Nucleosome formation potential of eukaryotic DNA: calculation and promoters analysis. *Bioinformatics*, 17:998–1010, 2001.

[88] R. Lopez. SRS - Sequence Retrieval System. Presentation Universidad Autonoma de Madrid. `http://www.pdg.cnb.uam.es/cursos/BioInfo2001/pages/-SRS/`, 2001.

[89] P. Luykx, I. V. Bajic, and S. Khuri. NXSensor web tool for evaluating DNA for nucleosome exclusion sequences and accessibility to binding factors. *Nucleic Acids Research*, 34:560–565, 2006.

[90] J. Madhavan, P. Bernstein, P. Domingos, and A. Halevy. Representing and reasoning about mappings between domain models. pages 80–86, 2002.

[91] I. Manolescu, D. Florescu, and D. Kossmann. Answering XML Queries on Heterogeneous Data Sources. In *VLDB*, pages 241–250, 2001.

[92] V. McKusick. *Mendelian Inheritance in Man. A Catalog of Human Genes and Genetic Disorders*. Baltimore: Johns Hopkins University Press, 1998.

[93] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity Flooding: A Versatile Graph Matching Algorithm and Its Application to Schema Matching. In *ICDE*, pages 117–128, 2002.

[94] R. J. Miller, D. Fisla, M. Huang, D. Kymlicka, F. Ku, and V. Lee. The amalgam schema and data integration test suite. `http://queens.db.toronto.edu/\thicksimmiller/amalgam/`, 2001.

[95] R. J. Miller, L. M. Haas, and M. A. Hernández. Schema Mapping as Query Discovery. In *VLDB*, pages 77–88, 2000.

[96] R. J. Miller, Y. Ioannidis, and R. Ramakrishnan. The Use of Information Capacity in Schema Integration and Translation. In *VLDB*, pages 120–133, Dublin, Ireland, 1993.

[97] www.dbis.informatik.uni-goettingen.de/Mondial.

[98] Multi-Node-Cluster. Setting Hadoop MapReduce on a Multi-Node Cluster. `http://hadoop.apache.org/common/docs/current/cluster_setup.html`.

[99] J. Munkres. Algorithms for the Assignment and Transportation Problems. *Journal of the Society of Industrial and Applied Mathematics*, 5(1):32–38, 1957.

[100] J. R. Munkres. *Topology*. Prentice Hall, Inc., 2000.

[101] K. G. Murty. An algorithm for ranking all the assignments in order of increasing cost. *Operations Research*, 16:682–687, 1968.

[102] N. F. Noy and M. A. Musen. PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. In *AAAI/IAAI*, pages 450–455, 2000.

[103] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: a not-so-foreign language for data processing. *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1099–1110, 2008.

[104] F. Ozsolak, J. S. Song, X. S. Liu, and D. E. Fisher. High-throughput mapping of the chromatin structure of human promoters. *Nat Biotechnology*, 25(2):244–8, 2007.

[105] A. H. P. Mork and P. Tarczy-Hornoch. A Model for Data Integration Systems of Biomedical Data Applied to Online Genetic Databases. 2001.

[106] L. Palopoli, D. Sacca, and D. Ursino. An automatic technique for detecting type conflicts in database schemas. pages 306–313, 1998.

[107] L. Palopoli, D. Sacca, and D. Ursino. Semi-automatic semantic discovery of properties from database schemas. pages 244–253, 1998.

[108] H. E. Peckham, R. E. Thurman, Y. Fu, J. A. Stamatoyannopoulos, W. S. Noble, K. Struhl, and Z. Weng. Nucleosome positioning signals in genomic DNA. *Genome Res.*, 17:1170–1177, 2007.

[109] L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernández, and R. Fagin. Translating Web Data. In *VLDB*, pages 598–609, 2002.

[110] L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernández, and R. Fagin. Translating Web Data. In *VLDB*, pages 598–609, 2002.

[111] R. Pottinger and P. A. Bernstein. Merging Models Based on Given Correspondences. In *VLDB*, pages 826–873, 2003.

[112] R. Pottinger and P. A. Bernstein. Schema Merging and Mapping Creation for Relational Sources. In *EDBT*, 2008.

[113] R.-H. Pusarla, V. Vinayachandran, and P. Bhargava. Nucleosome positioning in relation to nucleosome spacing and DNA sequence-specific binding of a protein. *FEBS*, 274:2396–2410, 2007.

[114] A. Radwan, L. Popa, I. R. Stanoi, and A. Younis. Top-k generation of integrated schemas based on directed and weighted correspondences. In *Proceedings of the 35th SIGMOD international conference on Management of data*, SIGMOD '09, pages 641–654, New York, NY, USA, 2009. ACM.

[115] A. Radwan, A. Younis, S. Khuri, M. A. Hernandez, H. Ho, L. Popa, and S. Shivaji. Biofederator: A data federation system for bioinformatics on the web. *Proc. AAAI Sixth Int. Workshop on Information Integration on the Web (IIWeb-07)*, pages 92–97, 2007.

[116] A. Radwan, A. Younis, P. Luykx, and S. Khuri. Prediction and analysis of nucleosome exclusion regions in the human genome. *BMC Genomics*, 9:186, 2008.

[117] A. Radwan, A. Younis, S. Srinivasan, and A. Gupta. MR-LEGOS: A Refined MapReduce Model. *Int. J. Cloud Computing - to appear*.

[118] E. Rahm and P. A. Bernstein. A Survey of Approaches to Automatic Schema Matching. *VLDB Journal*, 10(4):334–350, 2001.

[119] S. C. Satchwell, H. R. Drew, and A. A. Travers. Sequence periodicities in chicken nucleosome core DNA. *J. Mol. Biol.*, 191:659–675, 1986.

[120] J. Schug, W. P. Schuller, C. Kappen, J. M. Salbaum, M. Bucan, and C. J. S. Jr. Promoter features related to tissue specificity as measured by Shannon entropy. *Genome Biol*, 6:R33, 2005.

[121] E. Segal, Y. Fondufe-Mittendorf, L. Chen, A. Thastrom, Y. Field, I. K. Moore, J. P. Wang, and J. Widom. A genomic code for nucleosome positioning. *Nature*, 442(7104):772–8, 2006.

[122] R. Shaker, P. Mork, J. S. Brockenbrough, L. Donelson, and P. Tarczy-Hornoch. The BioMediator System as a Tool for Integrating Biologic Databases on the Web. In *Proceedings of the Workshop on Information Integration on the Web*, 2004.

[123] C. E. Shannon. A Mathematical Theory of Communication. *Bell System Technical Journal*, 27:379–423, 623–656, 1948.

[124] A. Sheth and J. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3):183–236, 1990.

[125] Single-Node-Cluster. Setting Hadoop MapReduce on a Single-Node Cluster. `http://hadoop.apache.org/common/docs/current/single_node_setup.html`.

[126] E. M. Smigielski, K. Sirotkin, M. Ward, and S. T. Sherry. dbSNP: a database of single nucleotide polymorphisms. *Nucleic Acids Research*, 28(1):352–355, 2000.

[127] B. Solutions. Bionavigator solutions. `http://www.bionavigator.com`, 2003.

[128] S. Spaccapietra and C. Parent. View Integration: A Step Forward in Solving Structural Conflicts. *Knowledge and Data Engineering*, 6(2):258–274, 1994.

[129] M. D. Stefano. *Distributed Data Management in Grid Environments*. Wiley-Interscience, 2005.

[130] M. Stonebraker, D. Abadi, D. J. DeWitt, S. Madden, E. Paulson, A. Pavlo, and A. Rasin. MapReduce and parallel DBMSs: Friends or foes? *Communications of the ACM*, 53(1):64–71, 2010.

[131] G. Stumme and A. Maedche. FCA-MERGE: Bottom-up merging of ontologies. In *IJCAI*, pages 225–234, 2001.

[132] A. Su and et. al. Large-scale analysis of the human and mouse transcriptomes. In *Proceedings of the National Academy of Sciences of the United States of America*, 2002.

[133] A. I. Su, T. Wiltshire, S. Batalov, H. Lapp, K. A. Ching, D. Block, J. Zhang, R. Soden, M. Hayakawa, G. Kreiman, M. P. Cooke, J. R. Walker, and J. B. Hogenesch. A gene atlas of the mouse and human protein-encoding transcriptomes. In *Proceedings of the National Academy of Sciences of the United States of America*, volume 101, pages 6062–6067, 2004.

[134] W. Sujansky. Heterogeneous Database Integration in Biomedecine. Methodological Review. *Journal of Biomedical Informatics*, 34:285–298, 2001.

[135] B. Suter, G. Schnappauf, and F. Thoma. Poly(dA.dT) sequences exist as rigid DNA structures in nucleosome-free yeast promoters in vivo. *Nucleic Acids Res*, 28:4083–4089, 2000.

[136] Y. Suzuki, R. Yamashita, S. Sugano, and K. Nakai. DBTSS, DataBase of Transcriptional Start Sites: progress report. *Nucleic Acids Res.*, 32 Database issue:D78–81, 2003.

[137] N. Taylor and Z. Ives. Reconciling while tolerating disagreement in collaborative data sharing. In *SIGMOD*, pages 13–24, 2006.

[138] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy. Hive: a warehousing solution over a map-reduce framework. *Proc. VLDB Endow.*, 2(2):1626–1629, 2009.

[139] A. A. Travers and A. Klug. Bending of DNA in nucleoprotein complexes. In N. R. Cozzarelli and J. C. Wang, editors, *DNA Topology and its Biological Effects*, pages 57–106. Cold Spring Harbor Laboratory Press, Cold Spring Harbor, NY, 1990.

[140] O. Udrea, L. Getoor, and R. J. Miller. Leveraging Data and Structure in Ontology Integration. In *SIGMOD*, pages 449–460, 2007.

[141] C. Vecchiola, S. Pandey, and R. Buyya. High-performance cloud computing: A view of scientific applications. *CoRR*, abs/0910.1979, 2009.

[142] Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou. Enabling public verifiability and data dynamics for storage security in cloud computing. *ESORICS*, pages 355–370, 2009.

[143] Y. H. Wang and J. D. Griffith. The [(G/C)3NN]n motif: a common DNA repeat that excludes nucleosomes. volume 93, pages 8863–8867, 1996.

[144] T. White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 2009.

[145] J. Widom. Role of DNA sequence in nucleosome stability and dynamics. *Quarterly Review of Biophysics*, 34:269–324, 2001.

[146] H. Xi, H. P. Shulha, J. M. Lin, T. R. Vales, Y. Fu, D. M. Bodine, R. D. G. McKay, J. G. Chenoweth, P. J. Tesar, T. S. Furey, B. Ren, Z. Weng, and G. E. Crawford. Identification and Characterization of Cell Type-Specific and Ubiquitous Chromatin Regulatory Strcutures in the Human Genome. *PLoS Genestics*, 3(8):1377–1388, 2007.

[147] L. Yan, C. Rong, and G. Zhao. Strengthen cloud computing security with federal identity management using hierarchical identity-based cryptography. *CloudCom '09: Proceedings of the 1st International Conference on Cloud Computing*, pages 167–177, 2009.

[148] C. Yu and L. Popa. Constraint-Based XML Query Rewriting for Data Integration. In *SIGMOD*, pages 371–382, 2004.

[149] G.-C. Yuan, Y.-J. Liu, M. F. Dion, M. D. Slack, L. F. Wu, S. J. Altschuler, and O. J. Rando. Genome-Scale Identification of Nucleosome Positions in S. cerevisiae. *Science*, 309(5734):626–630, 2005.