A NEW HIERARCHICAL CLUSTERING MODEL FOR SPEEDING UP THE RECONCILIATION OF XML-BASED, SEMISTRUCTURED DATA IN MEDIATION SYSTEMS

By

CHARNYOTE PLUEMPITIWIRIYAWEJ

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

2001

To my family, Sa-art, Kimsong, Suthisa, Chanchai and Sudaporn

ACKNOWLEDGMENTS

Over the years as a Ph.D. student at University of Florida, I have received support from many people. In particular, my mother, who lives in Thailand, and my brother always gave me their support. I consider myself extremely fortunate to work with Dr. Joachim Hammer, my supervisor, who introduced me to and encouraged me in the field of data warehousing and mediation systems. I am very proud to be a part of the IWIZ development team and to work with my colleagues, Renilton Soares de Oliveira, Anna Teterovskaya, Amit Shah, Rajesh Kanna, and Ramasubramanian Ramani. I would like to thank Karuna Ammireddy who helped me develop parts of the Result Evaluator and the Probe Query Generator modules. Aside from the individuals already named, I also benefited from discussions with other colleagues in the Database Research and Development Center, particularly in the EECOMS Project.

TABLE OF CONTENTS

page

LIST OF TABLES

LIST OF FIGURES

Abstract of Dissertation Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy

A NEW HIERARCHICAL CLUSTERING MODEL FOR SPEEDING UP THE RECONCILIATION OF XML-BASED, SEMISTRUCTURED DATA IN MEDIATION SYSTEMS

By

CHARNYOTE PLUEMPITIWIRIYAWEJ

August 2001

Chairman: Joachim Hammer
Major Department: Computer and Information Science and Engineering

This dissertation describes the underlying research, design and implementation for a Data Merge Engine (DME). Specifically, we have developed a hierarchical clustering model as a new solution to speed up the merging of similar and overlapping data items from multiple information sources. We use a tree-based heuristic algorithm for clustering data in a multi-dimensional metric space. Equivalence of data objects within the individual clusters is determined using a number of distance functions that calculate the semantic distances among the objects based on their attribute values. Because of the diversity of numbers of data items to be compared, we have developed a set of heuristics to appropriately reconcile data items. The experimental results show that our approach is more efficient and provides more accurate results when compared with other existing approaches.

Given the immense popularity of the World Wide Web (Web), we focus mainly on reconciling semistructured data. Specifically, we use the Extensible Markup Language (XML) as our internal data model for representing heterogeneous data. As part of our research, we have developed a comprehensive classification for schematic and semantic conflicts that can occur when merging data from related XML-based information sources.

The research proposed here is conducted within the context of the Integration Wizard (IWIZ) system, which allows users to access and retrieve information from multiple sources through a consistent, integrated view. To improve query response time, IWIZ uses a combined mediation/data warehousing approach to information integration.

CHAPTER 1
INTRODUCTION

During the last decade, the use of the World Wide Web (Web) has dramatically increased. The Web has become a very important information source that people can use to communicate, search for interesting information, and conduct business. In a sense, the Web can be viewed as a loosely connected information system that consists of millions of individual data sources. The data sources can range from simple data files to conventional database systems (e.g., relational and object-oriented ones). Integrating all these data sources and providing a single interface and data representation for users is a challenging problem since the sources are autonomous (i.e., independently managed), distributed (i.e., located in different places), and heterogeneous in nature (i.e., using different forms of representation).

Challenges

Turning the Web into a usable information system presents a major challenge. The challenge is caused by the heterogeneity. Two aspects of heterogeneity are technological differences (e.g. hardware, system software and communication systems) and schematic and semantic differences that exist among related data in different sources. To help solve the former problem, standard protocols and middleware components, e.g., CORBA, DCOM, ODBC, JDBC, etc., have emerged to simplify remote access to many standard source systems. To help solve the latter problem, so-called *integration systems* are introduced to overcome the schematic and semantic differences that exist among

cooperative data sources. The main problems related to the integration of data sources with different schemas and data semantics are the identification and the solution of conflicts between schema and data [34].

Although data integration problems have been investigated for several decades [12, 29, 35, 49], most work in this area has focused on integrating structured data (i.e., relational or object-oriented data). Recently, the concept of semistructured data, which can be represented using the eXtensible Markup Language (XML), was introduced [1]. A popular source of semistructured information is the Web in general and Web-based e-commerce in particular. Using XML to represent this semistructured data allows Web sites capture and represent more of semantics than using HTML. However, the data management involving semistructured data is relatively new. For example, the main concepts underlying the semistructured data model, specifically XML, are elements, subelements, and theirs attributes, whereas the main concepts underlying the relational data model are relations, attributes and attribute values. Because of the different characteristics of semistructured data, a new approach to integrate data sources containing semistructured data is needed.

Data-Integration Scenario

To show the need for an integration system, suppose an online shopping scenario where customers want to browse data on publications (e.g., books and articles) and related products (e.g., computer software). The data are stored in online bibliographies, digital libraries, online catalogs, and other e-commerce sites. For example, customers may want to access book information, pricing information and availability. They may

also want to access other online sources for related background information such as the customers' evaluation of the publications. This situation is showed in Figure 1.

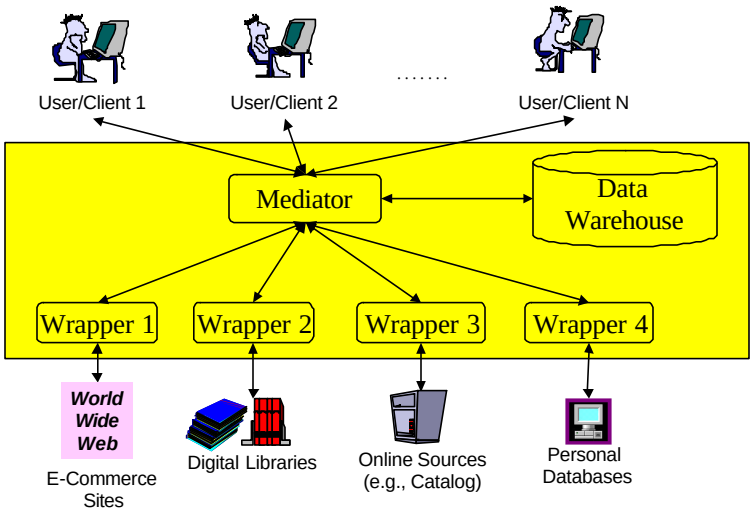

Figure 1: Conceptual view of an integration system accessing data from multiple, heterogeneous information sources. The integration system must be able to address data translation (source wrappers) as well as data fusion (mediator).

Typically, each source uses different tools and data modeling techniques to create and manage their data. This means the same concept, for example, the entity `software`, may be described by a different term and different set of attributes in different sources (e.g., `application`, `program` etc.). Also, there is no restriction on the underlying data model used to store and retrieve the source data. To enumerate a few possibilities for data representation, catalogs can be tables in the relational model, persistent objects in object-oriented databases, XML documents published on the Web, or flat files kept by legacy systems. Note that a consequence of the use of different data

models and systems to represent information is that sources provide different query capabilities [34].

In this online shopping scenario, users want to issue queries without being aware of which sources are available, where they are located, how they represent their data, and how they are individually queried. Instead, users prefer the illusion that all the data have been retrieved from a single large repository. In this scenario, such repository refers to the data warehouse storing frequently asked queries and their results.

Although integration systems do not restrict the way sources create and manage their data, they are built to provide users with transparent, integrated access to a wide variety of heterogeneous data. To accomplish this goal, the integration systems must deal with complex issues related to the fact that a real-world object can be represented differently in different data sources in terms of both structure and content of the object. For instance, one catalog listing computer books may represent author names as single character strings, whereas another source represents them as three distinct strings: first-name, middle-name, and last-name. In another example, the price of a book in one source may include shipping costs, whereas another source may separate base price and tax into different attributes [34]. In yet another example, a book in one source may consist of two authors, while the same book in another source may consist of only one author. The first example illustrates a difference in the structure of the data. The second example shows a difference in the domain of the data. The last example describes a difference in the content of the data. Differences in the structure and the domain of the data can be detected if we have knowledge about the schema of the data. Differences in the content of the data can be detected if we have domain knowledge. Detecting such

differences or *discrepancies* is important when we want to reconcile related data from multiple sources, as is the focus of this dissertation.

Goal of the Dissertation

To integrate the heterogeneous data sources, we are developing an integration system called Information Integration Wizard (IWIZ). The IWIZ is currently being developed at the Database Research and Development Center at the University of Florida [34]. The IWIZ allows end-users to access and retrieve information from various sources without knowledge about the location, API, and data representation of the underlying sources. At present, the sources are assumed to contain semistructured data in XML format. In the future, the system will provide access to information in other formats such as text files, relational databases, and image archives via wrapper components for various data models [64]. The IWIZ applies a hybrid data warehousing-mediation approach. The data warehouse is used as a cache storing the result of frequently asked queries. The IWIZ contains several components. Wrappers handle restructuring of the schema of the data sources. The mediator component supports the reconciliation of the data. The warehouse manager handles the users' queries and controls access of data warehouse.

This dissertation describes the design and implementation of Data Merge Engine (DME), a part of the mediator component of IWIZ system. The main tasks of DME are reconciling the data returned from underlying wrappers, providing a single, clean result to the users of the system. Inside DME, we apply a hierarchical clustering model to speed up the reconciliation of the data. Our model and underlying technologies are flexible and general enough so that they can be used in other integration systems that want to integrate semistructured data.

The rest of the dissertation is organized as follows: Chapter 2 provides an overview of the research that is relevant to this dissertation. Chapter 3 contains background information about XML-related technologies that are used in the IWIZ system. Chapter 4 presents the system design framework, the overall IWIZ architecture, the operational phases that are performed inside the mediator, and the place of DME in relation to other components. Chapter 5 describes one conflict classification for XML-based semistructured data. Chapter 6 explains the hierarchical clustering model and provides a framework for designing the clustering tree. Chapter 7 describes the results of a detailed qualitative analysis of our clustering model. Chapter 8 focuses on the implementation of DME. Finally, Chapter 9 concludes the dissertation with a summary of our accomplishments and an outline of issues to be considered in future releases of DME.

CHAPTER 2
RELATED RESEARCH

The research related to the work in this dissertation falls into the following five categories: Spatial access methods and similarity searches; semantic heterogeneities and conflict resolution; ontologies and knowledge representation; meta-search engines; and mediation and data warehousing system.

Spatial Access Methods and Similarity Searches

One of the problems in information integration is that a real-world object can be represented differently in different sources. To provide only one copy of an object acquired from the sources, we need to match all candidate objects and remove duplicate copies. Matching the objects can be viewed as a similarity search problem: Given an object, find the objects that are similar to the given object. Spatial access methods (SAMS) have been introduced to support similarity searches by indexing the spatial objects such as multidimensional points, lines, rectangles and other geometric objects [24]. Therefore, we can apply those methods for the element matching in our integration system. However, the goal of SAMS is different from ours. Spatial access methods aim to provide an index structure that is created at built-time and that can give a quick response to range queries and nearest neighbor queries. We do not use SAMS as an index structure, but as a way of resolving data conflicts, which are explained in detail in Chapter 5.

Several spatial access methods have been proposed, such as R-tree and its variants [7, 25, 32, 38, 61], TV-tree [50], X-tree [8], Metric-tree [65], MVP-tree [10], and M-tree

[13]. Those methods need a distance function that is used for comparing the data objects in the multi-dimensional space. The distance function measures the (dis)similarity between two objects. It is defined for all attributes and sub-objects of the considered objects. Dey et al. [20, 21] suggested several types of distance functions for different types of attribute values. Zobel and Moffat [87] provided a set of standard similarity measures that include combining and weighting functions.

In this research, M-trees are not used for indexing spatial data objects, but they are used for clustering data objects which are then reconcile. Like B-trees and R-trees, M-trees grow in a bottom-up fashion and are balance. The M-trees differ from B-trees in the fact that a node in M-trees is split when it is 100% full whereas a node in B-trees is split when its capacity reaches a certain threshold (usually when it is approximately 70% full). The M-trees differ from R-trees in the fact that M-trees use a sphere-shape routing-object to bound the objects that are close to one another in a particular space, whereas R-trees use a minimum bounding rectangle for that purpose. Another difference between M-trees and R-trees is the fact that M-trees compare objects in the Metric space, but R-trees compare objects in the Euclidean space.

In the Euclidean space, the distances between objects are uniformly defined. An object with *k* attributes can be viewed as a point in *k*-dimensional space. A transformation function is used to map the object to a point in the Euclidean space. Then, a distance function (e.g., Euclidean function) is used to compare the objects. In the Metric space, the distance between two objects is defined as relative without transforming the object. Let *x, y* and *z* be objects in the metric space and *d(x, y)* be the distance from object *x* to object *y*. The distance function *d* must satisfy the following properties:

(I) Symmetry: $d(x, y) = d(y, x)$

(II) Non-Negativity: $0 < d(x, y) < \infty$, where $x \neq y$ and $d(x, x) = 0$

(III) Triangular Inequality: $d(x, y) \leq d(x, z) + d(z, y)$

The X-tree, the TV-tree and the R-tree and its variants are suitable for indexing spatial objects in the Euclidean space. The Metric-tree, the MVP-tree, and the M-tree are suitable for indexing spatial objects in the Metric space.

Because of the characteristics of the semistructured data whose schema is flexible, we realize that matching elements in Metric space is more appropriate than matching elements in Euclidean space. Therefore, we evaluate those trees that compare elements in Metric space. Those trees belong to the family of metric-trees. Here, we will refer to "metric-trees" as trees that compare objects in the Metric space by applying a distance function that satisfies the symmetry, non-negativity and triangular inequality properties.

The shape of metric trees depends on a decomposition strategy used for adding elements into a cluster. Two basic strategies are *Ball Decomposition* and *Generalized Hyperplane Decomposition* [65].

The main idea of the Ball Decomposition is the following: Given a set of elements, arbitrarily pick one element from the set. We refer to that element as the pivot element. Then, calculate and keep the median of the distances from the pivot element to all other elements in the set. Next, partition the set of elements into two clusters. One cluster contains every element that is farther from the pivot element than the median. Another contains the remainder of the element in the original set. Finally, the process is applied recursively until all elements have been placed.

The main idea of the Generalized Hyperplane Decomposition is as follows: Given a set of elements, arbitrarily pick two different elements, for example x and y, from the set. The set is partitioned into three subsets. The first subset contains every element z

that is closer to *x* than to *y*. The second subset contains every element *z* that is closer to *y* than to *x*. The third subset contains the remainder of the elements in the original set.

We apply the tactic, namely picking one or more pivots, used in those two decomposition strategies as a framework for building our clustering tree. The tree is used to detect conflicts among semistructured data items and reconcile the items as a part of an integration system. Our integration system is explained in Chapter 4. Our decomposition strategy is described in Chapter 6.

Semantic Heterogeneities and Conflict Resolution

Information integration and sharing refer to the process of retrieving related information from multiple information contexts and using the combined information in a context (global view) that differs from the original context. Before one can share information in this way, four hurdles must be overcome: (1) understanding the meaning of the source data, (2) relating it to the global schema, (3) translating the values from the source context to the target context, and (4) merging related data. All of these hurdles are caused by heterogeneities between source and target, which exist at various levels of abstraction. Primarily, one is faced with heterogeneities at the system level (i.e., differences in the underlying hardware architecture, network protocol, operating system), at the data management level (i.e., differences in the data model, access commands), and at the semantic level (i.e., differences in the way related or similar data is represented in different sources).

We specified a new classification scheme for structural and semantic conflicts among XML-based data sources [56]. As one of our goals, we expect to resolve those conflicts in IWIZ. The main classes of conflicts are structural, domain and data conflicts. *Structural conflicts* arise when the schemas of the data sources that will be integrated

exhibit discrepancies. *Domain conflicts* arise when the schemas and domains of the data source that will be integrated exhibit discrepancies. *Data conflicts* arise when the semantic of the data sources that will be integrated exhibit discrepancies. The capability of resolving the structural and domain conflicts is hooked in the wrappers [64]. The capability of resolving the data conflicts is functioned in the mediator.

Most of the progress so far has been made in overcoming heterogeneities at the system and data-management level using translators and adapters. Kashyap and Sheth [39] proposed a formal model of similarity between objects in databases and classified conflicts based on such a model. Batini et al. [6] analyzed conflicts based on the ER model and propose a framework for the problem of schema integration. Kim et al. [42] promoted a Multidatabase language to resolve schematic conflicts among relational, object-oriented and Multidatabases. Dayal and Hwang [18] proposed a generalization technique to resolve discrepancies and inconsistencies found in multiple heterogeneous databases, including the use of an extended functional data model to model the databases. Kent [41] applied an object-oriented database programming language to resolve domain and schema mismatches. Lakshmanan et al. [47] and Miller [52] introduced a language derived from SQL to resolve schematic discrepancies. Siegel and Madnick [63] and Sciore et al. [60] used the metadata approach by attaching context information and resolved semantic conflicts based on the context.

Dey et al. [20, 21] proposed the probabilistic and distance-based decision models to resolve entity heterogeneity, which occurs when a real-world entity is presented differently in two databases in terms of its identifier. Specifically, Dey et al. applied the *Hungarian method* [45, 48, 55] to find the solution of the matching between two entity sets. The worst-case complexity of the method as a whole is $O(N^3)$ where N is the

maximum number between the numbers of the entities in those two sets. Even though the algorithm takes polynomial time, it is not practical for matching large data sets due to the high degree of complexity. Moreover, based on the experiment, the original decision model gives about 35% errors of matching two entity sets.

Like Dey et al., we aim to develop methodologies for efficient data merging. However, the run-time of our method is significantly faster. In our method, we use a clustering tree for resolving object reconciliation problem. Most of the run-time of our method was spent building the tree. Using the clustering tree, we found that the number of matching errors was between 1% and 5% depending on cluster size and the expected number of duplicates that can be found in the given data set.

Ontologies and Knowledge Representation

In order to create a basis for reasoning about the meaning of and relationships among concepts, we need a common metadata knowledgebase that is generally understood. It will provide a standard for the meaning of concepts. Such a knowledgebase is called *ontology* [31]. Ontologies are application-specific and dynamic: Application-specific because each contains most of the commonly used terms and their definitions for a particular domain; dynamic because the knowledgebase can grow as the need arises to contain new and updated terms. Ontologies have existed for many years and rules and guidelines have been worked out for their standardization. In addition, there exist many support tools for creating, editing, and sharing ontologies (e.g., ontology editors [26]). Ontologies are based on first order logic, making them powerful vehicles for reasoning about relationships. Not all definitions have to reside in a single ontology because there are various algebras for manipulating the contents of ontologies (see work on ontology algebras [82]). The Stanford Knowledge Sharing Group [44] is an excellent

source for comprehensive ontologies for many different applications. We envision that sharing ontologies will someday be as commonplace as exchanging documents or e-mail.

It is generally accepted that a domain-specific ontology is needed for data integration and data interchange. Several organizations (e.g., RosettaNet [58], CommerceOne [17], SchemaNet [59], XML/EDI [83] etc.) are working to specify standard ontology in the Business-to-Business (B2B) application domain. Farquhar et al. [26] introduced a web-based tool for ontology construction and provide an online ontology repository managed by the ontology server [43]. Maluf and Wiederhold [51] provided the basic concepts of constructing ontology using knowledge representations and the interoperation approach.

Mediation and Data Warehousing Systems

The query system used in our system, IWIZ, combines the mediation and data warehousing approaches. The data warehousing approach uses logically centralized, persistent storage to manage frequently accessed data and results of OLAP queries for faster retrieval. The mediation approach is used to support on-demand querying of the underlying sources in case the desired data are not available in the warehouse. The conceptual architectures of the mediation and warehousing systems are shown in Figure 2 and Figure 3, respectively.

In the mediation system, the information is extracted from the sources on-the-fly (i.e., only when queries are posed). For each user or application, the mediator receives an input query and decomposes it into subqueries to access the appropriate set of information sources. The mediator also merges the results of subqueries after the wrappers translate and filter the source information. After the merging, the mediator returns the query result to the user or application. Note that there is no persistent

repository to store the results of queries that may be posed multiple times; hence, it is inefficient in processing of frequently asked queries. This approach may incur the delay in query processing due to the traffic of the network during the integration process. However, the mediation approach is suitable when the sources change rapidly and when clients pose ad-hoc queries.

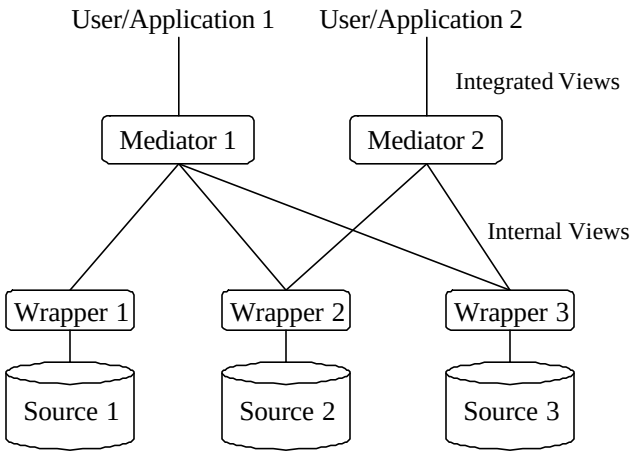

Figure 2: Schematic description of a mediation system architecture

In the data warehousing approach, the information of interest is extracted from the source in advance and materialized in the warehouse for future use. The wrappers translate and filter the source information. The integrator merges the relevant information from the sources and stores it in the warehouse. The warehouse provides a uniform access of information; hence, clients can create their views via the view manager without considering how data are managed in each source. The warehousing approach is suitable

when clients require high-performance of query processing, but do not require the most up-to-date information.

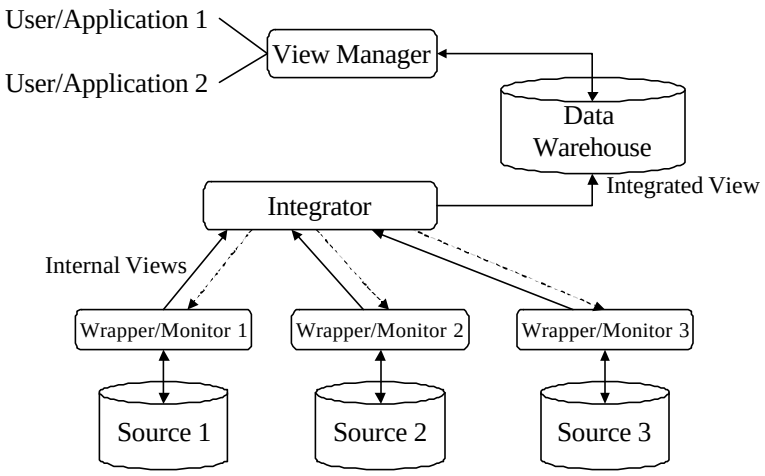

Figure 3: Schematic description of a data warehousing system architecture

In addition to the query system, other related research issues that need to be considered to design and build an integration system are global schema, common data models and representation, and mediation language.

Global schema. The purpose of the global schema is to provide a unified, integrated representation of relevant data from multiple sources. Using the global view, users and applications, henceforth referred to as clients, can indirectly formulate queries against the underlying sources without having to understand the actual source schemas [34]. An integration system can use a global schema in three different ways. The system can (a) have no global schema at all, (b) contain a single global schema, or (c) allow multiple global views, so-called federated schemas. If the integration system has a global

schema, that schema can be either derived from the sources, so-called the integrated schema, or predefined by clients and/or the system administrator. The TSIMMIS [12] mediation system has no global schema. Instead, a client defines a view based on the source schemas. The OASIS [57] and STRUDEL [27] warehousing systems allow multiple global views. The Infomaster [23], MIX [5], and WHIRL [16] are mediation systems with a predefined, single global schema. The MOMIS [9], WHIPS [46], and H2O [86] have a single global schema that is integrated from the source schemas. The IWIZ hybrid system has a single global schema that is a set of hierarchical concepts derived from a set of ontology. We assume that, in each application domain, a domain expert acting as a standards body has defined the ontology.

Two different approaches for defining the global schema are *global-as-view* and *local-as-view*. In the global-as-view approach, the global schema is defined as a view of local schemas. In the local-as-view approach, each local schema is defined as a view of the global schema meaning that the data in each source should be defined in terms of the global schema and no the other way around. The IWIZ system follows the local-as-view approach [34].

Common Data Model and Representation. The choice of data model and representation is one of the most important issues for researchers in the information integration area. Many proposals for data models are suitable for representing information in an integration system. The model can be in the class of structured or semistructured data. Structured data models include the ODMG's object model, the relational data model, the logic-based model as well as combinations of the above. Semistructured data models include the Stanford's Object Exchange Model (OEM), the graph-based data model, the XML-based data model and the like. The ODMG's object

model is used in Garlic [33], OASIS, MOMIS and WHIPS. Infomaster uses the Knowledge Interchange Format (KIF), which is a logic-based model. A frame logic-based (i.e., deductive object-oriented) data model is used in the COIN system [11]. The OEM is used in the TSIMMIS and Information Manifold [49]. The graph-based data model is used in STRUDEL and WHIRL. More recently, MIX [5] and IWIZ use the Document Object Model (DOM) [66] and XML [72] as the common data model and data representation, respectively. The reason for using XML and DOM is that they can capture and represent in a convenient manner a wide variety of loosely structured information that is available on the Web.

Mediation Language. The data model and data representation affect the choice of mediation language used in the integration system. There are many proposals for the language used in the mediator. Description logics and their variants are used in the SIMS project [2], Information Manifold, and TSIMMIS. The ODMG's object definition language and its variants (e.g., ODL , GDL, MOQL, and Ulixes) are used in MOMIS, Garlic, OASIS and Araneus [4]. The XML-QL [19] is used in MIX.

Many components and capabilities in IWIZ resemble those in MIX. Both systems focus on XML-based source information. They have a single predefined global view that clients can browse and query through a graphical user interface. They have wrappers and mediator that translate, restructure, and reconcile source information. However, important differences exist between those two systems. For example, MIX applies the mediation approach whereas IWIZ uses the hybrid data warehousing/mediation approach. In other words, views in IWIZ can be materialized in a persistent storage, i.e., data warehouse, while those in MIX are virtual. In addition, IWIZ focuses more on automating conflict resolution on the schematic and semantic levels with the help of existing information and

tools such as the ontology and standard dictionary. In MIX, a "mediator engineer" looks at the source schemas and information and pre-defines the single global view based on those schemas and information; hence, the resolution of potential conflicts is manually specified when the global view is built. In IWIZ, the global view is defined independently from the source schemas using the ontology structure. Therefore, the mapping of the global view and the source schemas is needed and it can be done automatically with verification from the system administrator. In other words, IWIZ reduces the system administrator's tasks of building the global view. In MIX, the mediator only contains modules for query processing, query planning and query optimization. In IWIZ, the mediator is capable of conventional query evaluation and can learn how to fuse, cleanse, and reconcile information.

Meta-Search Engines

A Meta-Search Engine (MSE) is a tool that accesses multiple (local) search engines [22, 84, 85]. It retrieves documents and ranks them according to their relevancy to the search phrase provided by the user. This meta-search engine is built on top of multiple local search engines (e.g., AltaVista, Yahoo, Infoseek).

The IWIZ, on the other hand, is an information integration system that provides an interface for users to access multiple heterogeneous sources: the user provides a query, and data is retrieved from one or more sources, merged and returned to the user. However, what constitutes a query in MSE is different from that in IWIZ. In MSE, a query (which is really more like a request and not a query) contains a list of keywords describing the contents of the desired documents (e.g., in form of Web pages) each of which is represented as a vector. In IWIZ, queries are represented using a formal query

language (e.g., XML-QL, SQL) and describe not only the desired contents but also the structure in which the results are to be returned.

In addition, MSE and IWIZ differ in the way that they treat and process the input data and query result. In IWIZ, we are focusing on sources that contain mainly semistructured data. The data are represented in the form of records (e.g., elements inside an XML document, records in an RDBMS, or individual facts in a text document) which can be fetched and returned to the user. In the case of MSE, the smallest level of granularity is a text document and the decision whether or not to return a particular document as part of the answer depends on the occurrence of search key values in the document. In IWIZ, a result is made up of one or more facts, whereas in MSE a result is made up of one or more documents that may or may not contain the desired facts.

Furthermore, the query results of IWIZ are represented using the schema or structure specified in the query. In other words, facts are integrated based on a user-defined schema. This is different from MSE whose query result is a ranked list of one or more retrieved documents. No integration is performed.

The IWIZ contains several components. The two important ones relevant to the work discussed here are the mediator, responsible for multi-source query processing, and one or more wrappers for source-specific data access. In a sense, the IWIZ mediator plays the role of the MSE, whereas wrappers play the roles of the local search engines. However, the way queries and results are processed inside the mediator and the MSE are different.

Given a user query, in IWIZ, the mediator decomposes the query into a set of mediated queries that will be sent to wrappers to access the requested data from the sources. Each query is specifically tailored to the results and query capabilities of the

underlying wrapper/source combination. The query decomposition is based on knowledge about the sources and the mediation specification that outlines the order and fashion in which the subqueries are to be processed. In MSE, given a user query, the MSE forwards the query to one or more local search engines to retrieve data in the form of documents. Note that the query is not decomposed.

In IWIZ, after a given query is decomposed, the mediator sends queries to the wrappers associated with the data sources to access the data objects. The schema or structure of the data from the source may be different from the user-defined structure specified in the query. Hence, the data from the sources needs to be restructured. In MSE, after selecting data sources, the meta-search engine invokes the local search engines associated with the selected data sources to access a set of useful documents. The local search engines may have to retrieve substantially more documents in order to guarantee that the desired documents are included in the result [84]. In addition, due to heterogeneities of sources, different local search engines may have different term-weighting schemes and different similarity functions. Therefore, the results from different local search engines may be incomparable and improperly ranked. In general, the search engines are autonomous and do not reveal the term-weighting schemes and the similarity functions they use [84].

In MSE, when the documents from different sources are retrieved, the meta-search engine compares the distances between the query and the retrieved documents (i.e., it computes the global similarity or score) and sorts them in a non-increasing order in a single ranked list which is returned to the user. In IWIZ, when the data objects from different sources are returned, the mediator merges the results into one integrated final result after cleansing the data (e.g., correcting inconsistencies) and eliminating duplicates.

After the data is integrated at the mediator, the result may need to be restructured again to conform to the user view specified in the query (since the global view used by the IWIZ mediator may not be identical to the view specified by the user).

A clustering technique is used for merging data. The data objects are clustered based on a distance function that is derived from several similarity functions that are provided for different types of data objects. The distance function specifies how likely it is that two data objects represent the same real world object. Each object can have sub-objects (attributes). The distance function between two objects of the same type is derived from the distances of corresponding sub-objects (attributes) and a set of weights associated with each sub-objects (attributes). The weight indicates how the sub-object (attribute) affects the clustering and can be adjusted manually by users (semi-automatic approach).

To make these assignments totally automated, we add a learning capability in the mediator. We can apply the same approach that is introduced in MSE and that is used for estimating a term-weight scheme, a local similarity function, and a threshold in each local search engine. The mediator sends probe queries to the sources. When the data are returned from the sources, the mediator cleanses, clusters and fuses the data using the distance function and preset values of the weight and other parameters. Then, users evaluate the merged result and provide feedback. Based on the feedback, the mediator adjusts the weight value and other parameters. Note that with this approach, different users can provide different feedback. This approach can be performed at built-time to accelerate the run-time query process.

In conclusion, both MSE and IWIZ mediator are tools to access data from multiple sources. However, they differ in (a) the format of the user query, (b) the

representation of the retrieved data sources, (c) the representation of the query result, and (d) the details of query processing and result merging. Although similarity functions are used in both MSE and IWIZ, in MSE, they are classified as global and local ones and are used for database and document selections, respectively. In IWIZ, the similarity function together with a set of parameters form a distance function that is used to determine how likely it is that two data objects represent the same real world concept. A learning capability is added into the IWIZ mediator to automatically assign an appropriate value for each parameter. A probe query is sent to the source to get a sampling data set. This approach is similar to that in MSE. Unlike MSE, the IWIZ mediator merges the results, returns it to the user, and waits for feedback. Finally, based on the feedback, the values of each parameter are adjusted.

CHAPTER 3
EXTENSIBLE-MARKUP-LANGUAGE-RELATED TECHNOLOGY

The work in this dissertation focuses on XML-based, semistructured data whose schema may vary. The concept of using XML to represent and manage data is relatively new. Many proposals involving XML-related components have been submitted to the World Wide Web Consortium (W3C) and are being revised. Some of those proposals have been approved and recommended by the W3C. With respect to W3C's recommendations, this Chapter is dedicated to provide the reader with brief introduction and background information on XML and its related technologies.

Extensible Markup Language (XML)

Extensible Markup Language describes a class of data objects called XML documents [72]. Data object instances are declared in an XML document (DOC) file. The structure of a document is defined by an optional Document Type Definition (DTD). The DTD can be either included in the DOC file or stored in a separate DTD file which is referred to by the corresponding DOC file. If they are in the same file, the DTD must be declared in the beginning of the document before the actual contents start (i.e., in the prolog portion). An XML document is well-formed if it meets requirements outlined in [36, 37, 72]. One way of verifying a well-formed XML document is using Internet Explorer 5.0. The document is valid if it is well-formed and associated with at least one DTD describing the structure of the document.

The XML was introduced as a way to capture and represent more of the semantics than HTML to facilitate the exchange of information. Goldman et al. [30] argue that XML should not be considered as a true data model since its specification does not define how to translate an XML document into data structures. Using this viewpoint, XML can be regarded as a language for representing information with data semantics. However, the World Wide Web Consortium (W3C) introduces the Document Object Model (DOM) [66] as a data model for XML. Moreover, the consortium provides working drafts of related XML components. For example, the eXtensible Stylesheet Language (XSL) [73] provides a framework for transforming XML and expressing stylesheets of a particular XML document. The XML Linking Language (XLink) [75] provides a framework for linking related XML documents, for linking the XML documents and metadata, and for linking multiple databases that reside in a location separate from the linked resources. The XML Pointer Language (XPointer) [76] provides a framework for addressing the internal structures of XML documents. The concept of XML Namespace [74] allows the identification and specification of elements and attributes across XML documents. The identification and specification is a collection of names that are used in XML documents as element types and attribute names. The XML Schema [78-80] provides a richer and more specific way for defining a database schema than the XML DTD Core structure. In addition, XML Schema is an extension of XML meaning that the underlying structure of a data schema that is specified by XML Schema is the XML DTD Core structure. However, by the time we started developing the IWIZ system, the tools (e.g., document parser) to manage XML Schema were not yet available.

Document Object Model (DOM)

As mentioned previously, DOM is introduced to be a data model for XML. We refer to DOM as a DOM Core structure. There are several extensions of DOM such as DOM Views [70], DOM Events [68], and DOM Styles [69] that build on the Core structure and that we are not using. At the time of writing this dissertation, there are three levels of DOM. The recommendations for DOM level 1 and 2 have been approved, whereas that for DOM level 3 [71] currently under revision. Basically, DOM level 3 builds on DOM level 2 [67], which builds on DOM level 1 [66]. The DOM recommendations mainly contain a set of platform- and language-neutral interfaces to create and manipulate the structure and contents of a document. The document that we refer to can be HTML, XML or any other document that contains semistructured data that can be parsed into the DOM structure.

The atomic DOM structure is a `Node` object. The `Attr`, `Element`, `Document`, `Text` and other objects are special types of `Node` object. The `Document` object represents the entire document which can be considered a hierarchy of `Node` objects. The `Element` object represents an XML element which can contain other elements including associated attributes. The `Attr` object represents an XML attribute of a particular element. The `Text` object represents either an attribute value or the data value for a particular element.

Query Language for XML

The availability of large amounts of data on the Web creates a need for tools for querying, extracting, transforming, and integrating data from documents [19]. The Structured Query Language (SQL) and Object Query Language (OQL) are standard

query languages for manipulating data from relational databases and object-oriented databases, respectively. For semistructured database, several research groups have proposed query languages such as Lorel [1, 30] and YaTL [14, 15]. However, each of those languages was originally designed for manipulating only data that was represented in a data model, which is different from the XML-based data model. Because of the popularity of using XML to represent data on the Web, the functionality of manipulating XML data was added to those languages. Since then, many more query languages for XML data such as XML-QL [19], XSLT [81] and XML Query Algebra [77] have been proposed.

Although Lorel, YaTL and XML-QL were developed independently by different research groups, they have similarities in their design approach and similarities of capability for querying XML documents. The similarities and differences among those three query languages are outlined in [28]. Like SQL, XML-QL has a SELECT-WHERE construct which comes in form of WHERE and CONSTRUCT clauses. The WHERE clause has two parts. The first part specifies input data, its schema, and location. The second part, which is optional, indicates a set of filters or conditions. The CONSTRUCT clause identifies the structures of user-defined views. Inside both WHERE and CONSTRUCT clauses, the schema is declared as the hierarchy of tags, the fundamental component in XML. Since XML-QL is more syntactically related to XML than Lorel and YaTL, IWIZ follows syntax of XML-QL for the design of the system query language.

XSLT was originally designed for use as part of XSL, which is a stylesheet language for XML [73, 81]. One can view XSLT as a query language for XML because

it can transform one XML document to another XML document. XSLT uses the formatting vocabulary included in XSL to describe how the document is transformed into another XML document. The transformation is specified in the form of rules. Not only do users need to know the schema or structure of view and input data, but they also need to understand the predefined, formatting vocabulary, like HTML, to create an XSLT rule. Users who understand only the schema of the input data can easily and quickly write an XML-QL query rather than XSLT rules to generate their own view.

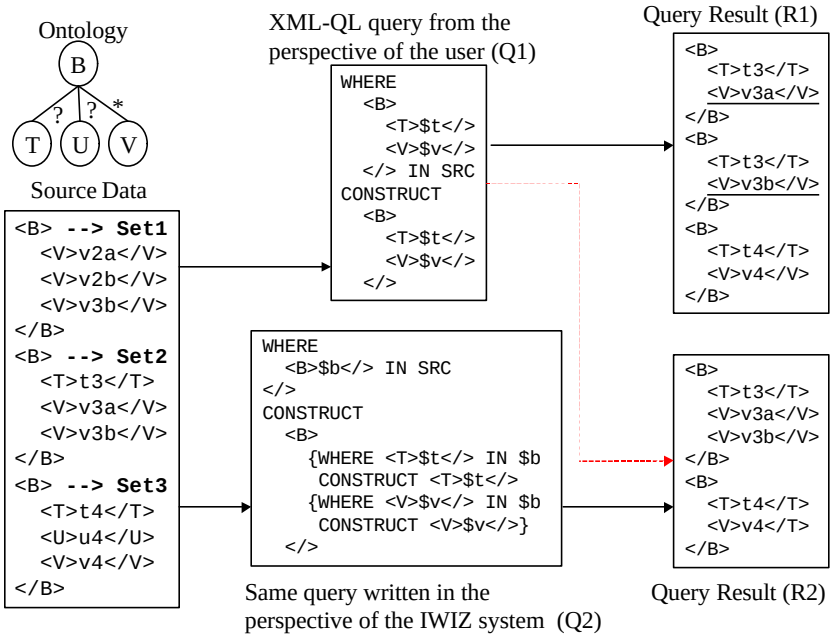

Figure 4: Two different representation of the same XML-QL query.

Recently, the XML Query Algebra (QA) was introduced as a formal basis for an XML query language [77]. Theoretically, QA is as powerful as all other query languages mentioned above. Unlike XML-QL, QA does not use XML, but a mathematics-like notation for expressing query. Since QA is only a recommendation to the XML

standards committees at this point, no QA query processor has been implemented at the time of this writing.

As we mentioned earlier, IWIZ follows the syntax of XML-QL for the design of the system query language. Semantically, however, IWIZ's query language is different from XML-QL. To show the differences in query processing between XML-QL and IWIZ, a scenario is given in Figure 4. Suppose an XML source contains three sets of elements of type B whose schema can be described as follows: Each element of type B consists of zero-or-one element of type T, zero-or-one element of type U, or zero-or-more elements of type V. Let us assume that, the first set contains one element of type B that has three elements of type V; the second set contains one element of type B that has one element of type T and two elements of type V; the last set contains one element of type B that has one element of type T, one element of type U, and one element of type V. Assume further that users want to query for elements of type B in the second set without changing the contents of each element of type B. If users write a single, non-nested query (refer to Q1 in Figure 4) to the XML-QL processor, the processor will return the result (refer to R1 in Figure 4) containing a Cartesian product of subelements of the root element of type B from the second set of the source data (shown at the bottom left corner of Figure 4). The result R1 is incorrect in IWIZ environment in which each data source may contain partial information. To get the correct result (refer to R2 in Figure 4), users need to write the nested query (refer to Q2 in Figure 4) instead. Note that the more hierarchical the structure of the subtree root element, the higher the degree of nesting of the query. In IWIZ, we simplify the task of our users by allowing them to send the non-nested query (Q1) to IWIZ. The result (R2) corresponding to the nested query (Q2) is

returned. To make this happen, IWIZ has its own query processor which is extended from the original XML-QL processor. IWIZ's version of the query processor is partitioned into three modules each of which is encapsulated in different IWIZ components (Warehouse Manager, Mediator and Wrapper.) The detail of IWIZ architecture is provided in Chapter 4.

Another goal of IWIZ is to shield users from source details (e.g., API, schema, vocabulary). Therefore, the IWIZ query processor needs to collect all partial results from the sources and merge the results, whereas XML-QL processor will access only the full result (i.e., elements that satisfy the input query) from the sources, and ignore partial results at the sources. As shown in Figure 4, with respect to query Q1, XML-QL will not return the element of type B in the first set since the element contains incomplete information (i.e., the user asks for elements of type T and V in element of type B, but the first set contains only elements of type V.), whereas IWIZ will internally access from the source such incomplete elements which can possibly be merged to other incomplete elements that come from other sources.

Parser for XML

Extensible Markup Language is a language that is used for representing data objects which are declared in an XML document (i.e., text file). Like other languages (e.g., HTML, VRML, C, Pascal, etc.), XML needs a parser to parse the objects declared in the document and store those objects into a data structure so that we can easily manipulate them. The parsers for XML can be categorized into two groups, SAX (Simple API for XML) and DOM, depending on the underlying data model the parser

uses. Both SAX and DOM parsers provide a set of APIs for manipulating XML documents.

The most important difference between SAX and DOM is that DOM represents a document as a tree of nodes, whereas SAX represents a document as a sequence of events (i.e., it calls a handler function as each chunk of XML syntax is parsed and recognized). Because of the event-based approach, a drawback of SAX is that it does not support random-access manipulation of documents as is supported by DOM, which allows random-access to any node in the document tree. On the other hand, SAX allows users to create their own custom object model by constructing a class that listens to SAX events, whereas DOM already provides an object model for users.

In IWIZ, we are using a DOM parser to parse XML documents, since it provides a set of utility routines to manipulate the objects declared in the documents. In addition, the DOM parser is also used inside the XML-QL query processor that is invoked by each IWIZ component. Two DOM parsers are used in IWIZ: IBM's and Oracle's DOM parsers. Both provide the same basic APIs that has been recommended by W3C. However, by the time we started building the IWIZ system, Oracle's parser provides a set of APIs that allows us not only to parse an XML document but also to individually parse an XML DTD file. In IWIZ, it is very important to have such functionality since we have to annotate the ontology which is defined in an XML DTD file. Therefore, Oracle's parser is used in each IWIZ component, as opposed to XML-QL query processor which uses a parser from IBM.

CHAPTER 4
INFORMATION INTEGRATION FRAMEWORK

A basic problem of integrating heterogeneous data sources is that a real-world object is represented differently in different data sources in terms of both structure and semantics of the data. To resolve this problem, we are developing an integration system which contains components to resolve the conflicts that can occur when two data sources containing related and overlapping information are accessed simultaneously. The system, called *Integration Wizard* (IWIZ), combines existing mediation and warehousing approaches to improve performance and functionality of traditional integration systems. The data warehousing approach uses a logically centralized, persistent data store for frequently accessed data and results of OLAP queries to speed up retrieval. The mediation approach supports on-demand querying of the underlying sources in case the desired data is not available in the warehouse.

The design of the system is based on the following two premises. First, a user can browse the global ontology, which is defined by a domain expert as a set of hierarchical concepts for a particular application, without knowing where the actual data are stored and in which native format they are represented. The user can query the data based on the schema in the global ontology. We envision that the users will put together the structure and associated data constraints that specify the desired result by pointing and clicking through the concept hierarchies in the ontology using a GUI.

Second, the result for a user query is stored in a data warehouse to provide a) persistence, b) efficient support for frequently asked queries including the ability to let the user refine the result further without having to go back to the sources, and c) automatic maintenance of the result in light of changes to the sources. We envision that sources can come and go which means that there will always be a lag-time between what the system "thinks" is available and what is indeed available.

Integration Wizard System Architecture

An overview of the IWIZ system is shown in Figure 5. System components can be classified into two categories: Storage and control. Storage components include the sources, the warehouse, and the metadata repository. Control components include the wrappers, the mediator, the warehouse manager, and the querying and browsing interface. In addition to those components, there is a set of information flow in the system not shown in the figure. This information flow refers to global schema, query and data.

We relate the *Global schema* (GS), for a particular application, to a set of hierarchical concepts derived from the system ontology. We assume that, in each application domain, a domain expert is responsible to define the ontology.

The system manipulates the data through a set of *queries*. There are four types of query: User query, warehouse query, mediated query, and restructured query. The users generate user queries through the user interface. The warehouse manager creates warehouse queries to access data from the warehouse and to request the mediator to collect data from the sources. The mediator produces mediated queries to request data

from the sources via the restructuring engine. Inside each wrapper, the restructuring engine generates restructured queries to access the source.

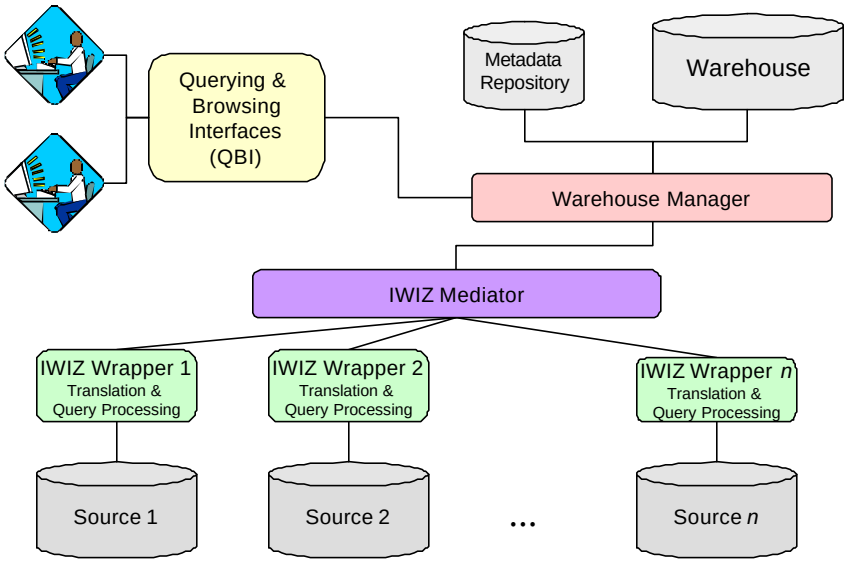

Figure 5: Schematic description of the IWIZ architecture and its main components

In IWIZ, *data* are represented as a set of XML documents. The name of the type of data is based upon which controlling component produces the data. The sources provide the requested source data to be integrated. The wrappers translate and restructure the source data into the restructured data and provide them to the mediator. The mediator produces the mediated data to answer a warehouse query by fusing the restructured data and cleaning the redundant and overlapping data. The query result refers to the final answer for a given user query.

Now we are ready to identify the components in IWIZ starting from storage to control components:

Sources. Ideally, the sources can be structured (e.g., relational and object-oriented), unstructured (e.g., text document), or semistructured (e.g., Web data). However, XML-based, semistructured sources are the current focus of this research. We refer to the XML-based, semistructured information as the underlying source information. We assume that all data are represented using XML together with the corresponding DTD which explains the schema of the data. If the underlying sources have no explicitly defined DTD, one must be generated.

Warehouse (WH). It is a repository that stores user-query results. It provides fast access for frequently asked queries and persistence for XML documents. To warehouse data items that are declared in an XML document, a data model is needed. Although Document Object Model (DOM) is the data model that is more related to XML data than the relational or object-oriented data model, there are few systems for persistently storing DOM objects. Therefore, an alternative is to use an existing database management system, such as a Relational Database Management System (RDBMS), an Object-oriented Database Management System (ODBMS), or an Object-Relational Database Management System (ORDBMS). We are using the Oracle's XSU (XML SQL Utility for Java) to persistently store data items. The XSU provides a Java API to store and access XML documents using Oracle8i's relational database manager.

Metadata Repository (MR). It serves as a persistent repository for storing auxiliary data such as global schema, information about sources, internal information, and restructuring and mediation specifications. To manage the metadata, the management system used to maintain the WH can be applied.

Wrapper. Its tasks are to 1) map underlying data model used in the underlying sources into a common data model used in the integration system, 2) map the structure of data represented in the underlying sources, i.e., the source schema, into semantically equivalent concepts defined in the ontology, 3) take a mediated query and transform it to a restructured query to access the underlying source with the help of the mapping generated in the first task, and 4) restructure the underlying source data.

Mediator. Its tasks are to 1) collect metadata and information about underlying sources and store them into the metadata repository, 2) transform a warehouse query into a set of mediated queries and send them to corresponding wrappers, and 3) merge restructured data and return them to users via the warehouse manager.

Warehouse Manager. Its tasks are to 1) take a user query and analyze whether or not the requested data are in the warehouse, and 2) return a result that can be either found in the warehouse or acquired from the mediator to users via the Querying and Browsing Interface.

Querying and Browsing Interface. Its tasks are to 1) provide a graphical user interface to allow users to query and browse information in the system, and 2) show the query result of a user query in easy-to-understand form.

Information Reconciliation

In the previous section, we provided a design framework for an information integration system and explained the functions in each component inside the system. In this section, we present an operational approach to integrate heterogeneous data sources. Our approach is applicable not only to our integration system, IWIZ, but also to other

integration systems having similar components. Therefore, by default, we will refer to IWIZ as a generic integration system unless we specifically indicate otherwise.

We differentiate our integration system into three operational phases: Metadata Construction (MC), Data Access (DA), and Metadata and Warehouse Maintenance (MWM). They are executed at build-time, run-time and maintenance-time, respectively. The main focus of the MC phase is to analyze the schema of the underlying source data and to collect all information about the sources. The main focus of the DA phase is to analyze information about the sources in order to accelerate and automate the restructuring and merging of similar and overlapping information. Finally, the main focus of the MWM phase is to update in periodic intervals the information about the sources and revisiting the process of analyzing the source schema and to revise the process of analyzing the source schema, when the data and the sources are changed.

The MC phase is very important since we envision that all metadata that we constructed in this phase will automate and accelerate the integration process in the DA phase. This approach is different from other approaches in that we emphasize the significance of the built-time activities. We refer to the metadata collected in the MC phase as a set of specification information. However, to decide what kinds of metadata should be constructed at built-time, we need to understand the overall integration process at run-time. In addition, we need to be aware of how to manage the metadata so that it can be easily accessed, updated and manipulated at both run-time and maintenance-time.

Note that, inside each system component (e.g., Wrapper, Mediator, etc.), those three phases operate sequentially. However, if we consider the entire system, they are not totally sequential. For example, at built-time, the mediator is responsible for collecting

all information about the sources and the data residing in the sources. To do so, the mediator may need to send sample queries to the wrappers to get the data to be integrated. Each wrapper can automatically process the query as long as it has the specification that must be constructed beforehand. That means the MC phase of the mediator includes executing the specification in each wrapper against the query from the mediator to get the data that has been restructured and ready to be analyzed.

The focus of this research is to develop more efficient and accurate algorithms for reconciling data inside the mediator. Therefore, the next sections will emphasize the details of each operational phase in the mediator. For the whole system, the overview of each operational phase can be found in [34].

Metadata Construction

As a whole, the primary goal of the Metadata Construction (MC) phase is to generate two different sets of rules for converting source data from its native representation into the integrated format specified by the ontology (recall that we refer to the ontology as a set of concept hierarchies for a particular application): (a) for each source, a source-specific restructuring specification for carrying out the schema alignment, and (b) a single mediated specification for carrying out the query processing and data merging. The rule sets for both activities are generated into subsequent steps and will be used as inputs to the restructuring engine (a module inside the wrapper) and the mediator respectively of the data access phase at run-time.

The ultimate goal of the MC phase is to populate the integrated schema with relevant data and store them into the data warehouse for future fast access. We envision that there is a set of queries whose (partial) results can be derived from a more general query (i.e., the answer of the query with a constraint is a part of the answer of the query

without the constraint). For example, the queries, "given ISBN number of a book, show me all information about the book." and "given an author of a book, show me the name and ISBN of the book." can be answered by projecting the answer of the query "show me all information about the book." We can see that the last query is more general than the first two queries. We use the warehousing approach to reduce the time of processing the query and access the data at run-time.

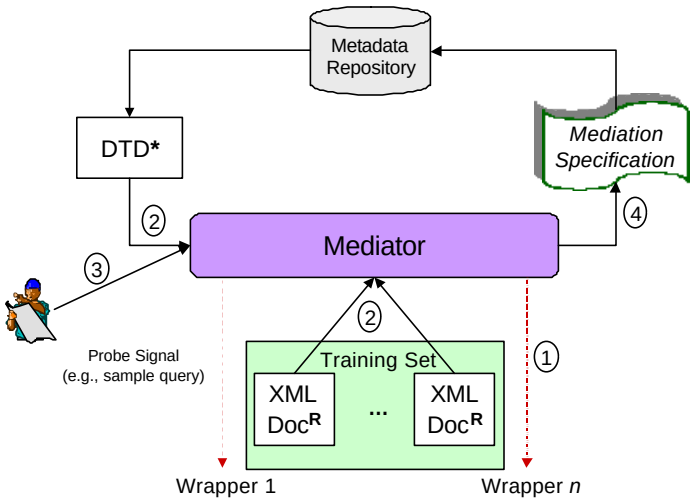

Figure 6: Metadata construction inside the IWIZ mediator

Initially, we assume the warehouse is empty (i.e., no integrated data is stored in the warehouse). We also assume that there exists a global schema derived from a set of one or more ontologies. The global schema describes a set of concept hierarchies of the application area of interest. It is stored in the Metadata Repository (MR). For example, suppose we have configured IWIZ to access bibliography data on-line. We assume that there is an ontology containing concepts of book, article, proceedings and other type of publications. We refer to a global schema as a set of structural definitions of each concept

in the ontology. The global schema is sometimes also referred to as target schema during the built-time restructuring and merging process.

Figure 6 shows the processes for the Metadata Construction phase of the mediator. They can be explained as follows: First, the mediator sends a set of sample queries to each wrapper to poll sample data set as well as the information about the sources such as the availability and reliability of each source (i.e., which underlying source contains which subsets of the hierarchical concepts defined in the ontology, and which source is more reliable if the sources contain overlapped data). Second, each wrapper returns a training set of the *restructured* XML documents (DOC^R in Figure 6). Based on the global schema (DTD* in Figure 6), the mediator analyzes those documents and resolves the conflicts among them by merging the documents. The outcomes of this process are the clustering-decision tree (see Chapter 7) and a draft of the mediation specification. Next, the system administrator verifies the draft of the mediation specification and the hierarchical clustering tree. Finally, the approved mediation specification and the hierarchical clustering tree are stored in the metadata repository for future use.

Data Access

As a whole, the goal of the data access phase is to allow the end-users to browse the concept hierarchies and access the integrated data that are either stored persistently in the data warehouse or requested from the underlying sources without knowing which source(s) they come from and in which native format they are represented. In this situation, the data warehouse (DWH) and the metadata repository (MR) have been set up during the built-time. The MR stores the global schema information, the restructuring

specification and the mediation specification. The system will use such information to process the user query and give the result to the user.

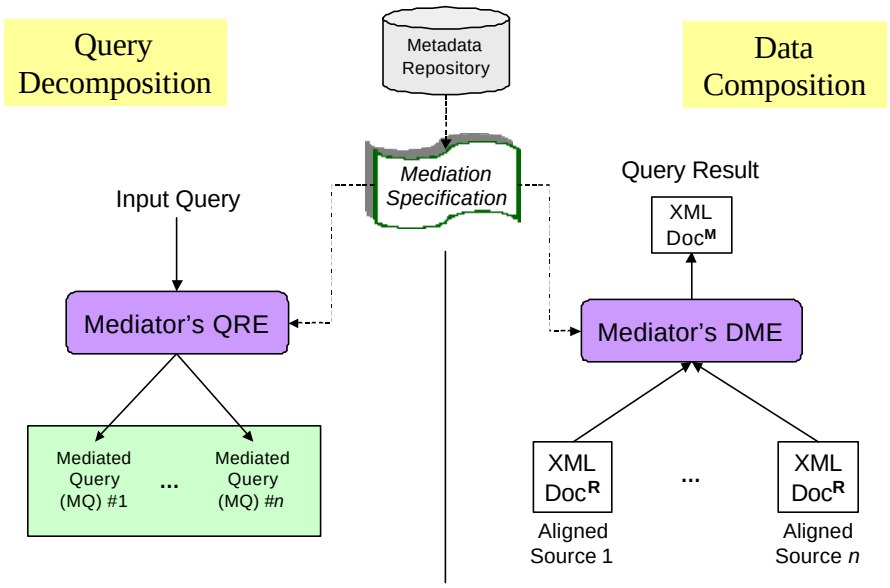

Figure 7: Data access inside the IWIZ mediator

Figure 7 shows two processes in the Data Access phase of the mediator. Those two processes are Query Decomposition (QD) and Data Composition (DC) each of which is performed inside two sub-components of the mediator, namely *Query Rewrite Engine* (QRE) and *Data Merge Engine* (DME).

In the QD process, the Query Rewrite Engine inside the mediator takes an input query from the warehouse manager. Then, based on the mediation specification, the mediator decomposes the input query into a set of mediated queries (MQs). In the mean time, it also generates the query plan that specifies the sequences of join operators for combining partial result from the individual sources. Finally, each MQ is sent to the

wrappers that will restructure the query and process the query to access the underlying sources.

The DC process starts when the result of each MQ is returned from the wrappers as a set of restructured XML documents (DOC^R). By following the mediation specification and the query plan given by the Query Rewrite Engine, the Data Merge Engine fuses those documents, and resolves the conflicts caused by the overlap of similar and related that data. The outcome of this process is the query result represented as a merged document (DOC^M).

The details of our algorithm for reconciling data inside the mediator are provided in Chapter 6. A classification of conflicts that must be resolved is given in the next Chapter.

CHAPTER 5
CONFLICT CLASSIFICATION FOR XML-BASED SEMISTRUCTURED DATA

In the previous chapter, we proposed an operational approach to accomplish the information reconciliation in the mediator and its data merge engine. In this chapter, we classify the conflicts that can occur among XML-based, semistructured data into three main classes: Structural, domain and data conflicts. *Structural conflicts* arise when the schemas of the data sources that will be integrated exhibit discrepancies. *Domain conflicts* arise when the schemas and domains of the data sources that will be integrated exhibit discrepancies. *Data conflicts* arise when the semantics of the data sources that will be integrated exhibit discrepancies. The class of structural conflicts includes generalization conflicts, aggregation conflicts, internal path discrepancy, missing items, element ordering, constraint and type mismatch, and naming conflicts for elements and attributes. The class of domain conflicts includes schematic discrepancies, scale or unit discrepancies, discrepancies with precision, and data representation conflicts. The class of data conflicts includes the ID-value conflicts, missing data, incorrect spelling, and naming conflicts for element contents and attribute values. In our integration system, resolving structural and domain conflicts is the responsibility of the wrappers and has been described in a related research effort [53, 64]. Resolving data conflicts is the responsibility of the mediator. A detailed treatment of structural and domain conflicts among related XML data sources is provided in our technical report [56].

The basic problem that must be resolved in the mediator is to identify the same real-world object that is differently (and partially) represented in the sources. To illustrate the problem, consider the two `Book` elements shown in Figure 8.

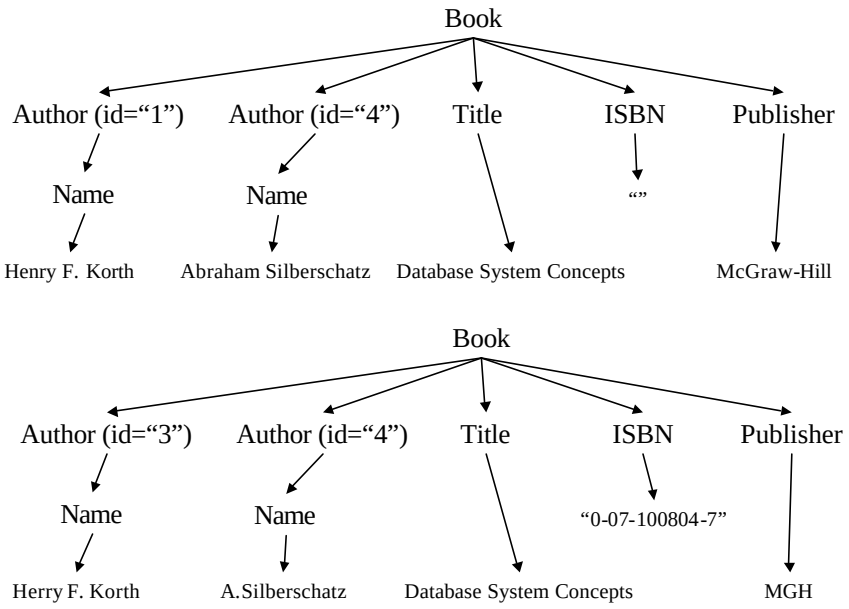

Figure 8: Two `Book` Elements from Two Different Sources

Both `Book` elements are from two different XML-documents (i.e., sources). In the real world, they represent the same book. Therefore, when a user queries for this book, the system should return only one copy of this element in the query result. However, mapping these two elements may not be straightforward, since both elements contain several conflicts. The figure shows several types of data conflicts. The `Name` of the first `Author` of the bottom `Book`, "Henry", is misspelled as "Herry"; hence, misspelling conflict occurs. The `Name`s of the second `Author` of each `Book`, "Abraham Silberschartz" and "A. Silberschartz", represent a synonym conflict since they

represent the same person but are expressed in different values. The ISBN for the top Book is missing; a missing item conflict is created. The Publisher, "MGH", of the bottom Book is an acronym for the Publisher, "McGraw-Hill", of the top Book; an acronym conflict is created. Last but not least, all first Author values of each Book refer to the same person, but they have different id values; an ID-value conflict is created.

Note that both Book elements have the same underlying structure. In other words, each Book contains the subelements Author, Title, ISBN and Publisher. This is the result of the schema restructuring that is performed by IWIZ wrappers which transform elements whose structure is defined in the source schema into elements whose structure is defined in the target schema. Consequently, all elements provided to the mediator have the same structure as the one defined by the global schema. However, in the global schema, the substructure of an element may also be associated with some constraint e.g., exactly-one, one-or-more, zero-or-more, and zero-or-one. For example, the global schema defines the structure of the Book element to contain at least one Author. Different Book elements represented in an XML document may have different number of Author subelements. One Book may have two Author subelements, whereas another Book may have only one Author subelement. Thus, the missing item conflicts occur.

In Figure 8, we only show the data conflicts between sub-elements (e.g., Author, ISBN, and Publisher) of a particular element (e.g., Book). The data conflicts can occur between attributes associated to a particular element as well. For example, in Figure 8, if both sources represent a Publisher as an attribute of Book

element, instead of a sub-element of `Book` element, the acronym conflicts still exists. In conclusion, all data conflicts arise when contents of two elements or attributes exhibit discrepancies.

In the next chapter, we describe a new hierarchical clustering model as a part of resolving the data conflicts that we have mentioned. The model considers the case that an element in one source partially agrees (on common attributes, and/or subelements) with other elements in the other sources.

CHAPTER 6
HIERARCHICAL CLUSTERING MODEL

To resolve data conflicts such as those described in the previous chapter, we have developed a new hierarchical clustering (HC) model. The model considers the case that an element in one source partially agrees (on common attributes and/or subelements) with other elements in the other sources. In the example in Figure 8, both `Book` elements agree on all subelements except `ISBN` whose value is missing from the top `Book` instance. In addition, the HC model does not assume that elements from the same source must be distinct (i.e., we allow the case where elements occurring in the same source can represent the same real-world element).

Generally speaking, the hierarchical clustering model contains a set of distance functions and a clustering tree. The distance functions define the closeness of two (XML) elements based on their subelements and attribute values. The clustering tree is the hierarchical structure of a set of clusters. In the following sections, we will explain the framework of choosing the clustering tree, defining the distance functions, as well as the clustering strategies that we have developed for carrying out the data reconciliation.

A Framework for Clustering Trees

The concept of clustering elements is not new, especially in the spatial database and data mining communities. However, it has never been applied to data integration. Many clustering trees have been proposed such as the R-tree and its variants, the Metric-tree and its variants, as well as the classification tree. For our own hierarchical clustering

model, the question whether to construct a new tree or choose an existing tree originally applied in other research areas was one of the driving forces behind this dissertation work.

The clustering tree is a tree that contains a set of clusters. Each cluster contains a set of elements that are related to each other. The most suitable tree structures for this dissertation work are those that have a small *run-time complexity* for the construction phase and that result in a low *percentage of the errors* during element matching. By small run-time complexity, we mean that the tree should be built in polynomial degree time. By low percentage of errors during matching, we generally aim to 10% or less.

There are two types of errors when matching elements. A *False alarm* arises when the integration system indicates that two elements match, although they are unrelated in the real world. A *false dismissal* arises when two elements, which are related in the real world, are mismatched. The number of false alarms is counted from the number of comparisons between two elements, which represent different real-world entities, in the same cluster. The number of false dismissals is counted from the number of comparisons between two elements, which represent the same real-world entities, from different clusters. The total number of errors is obtained by accumulating the number of false alarms and false dismissals. Finding an appropriate trade-off between false alarms and false dismissal is a research problem.

As we mentioned, constructing a new tree or choosing an existing tree originally applied in other research areas is another important research problem. The following defined our framework in which we have designed our clustering tree:

Distribution of Data

Data can be uniformly distributed or highly correlated. Uniform distribution means that the data are distributed evenly across all dimensions. High correlation means that the data occupy only some subspace of a high-dimensional space. In the data integration domain, the data are non-uniformly distributed but may be highly correlated. However, related data elements may occupy different subspaces of the high-dimensional space, and unrelated data elements may occupy the same subspace in the high-dimensional space. For example, in Figure 8, two `Book` elements are related since they represent the same book, but one has an `ISBN` number, whereas the other does not. In addition, let us assume that there is another element of type `Proceedings` that has the same structure as the `Book` element except `Proceedings` does not have an `ISBN`. When we map those elements into the high-dimensional space (each dimension represent the subelement of the elements), the first `Book` element having no `ISBN` may be closer to the `Proceedings` element than the second `Book` element containing the `ISBN`. Thus, in the data integration domain, the distribution of the data is correlated due to overlapping of data values.

Types of Data

A data object is considered to be a spatial data item when it can be mapped to a point in a multi-dimensional space. In the data integration domain, data objects may not be easily mapped to a point in a multi-dimensional space. For example, when mapping two text values, one normally uses edit distances as a measure of how closely related they are, instead of transforming those values into the points in multidimensional space.

Characteristics of the Input Data Set

Some of the existing tree structures, such as the X-tree, work very well for dynamic data sets because the tree is robust under insertion (i.e., it reorganizes the directory to avoid repeated disk accesses for searching). In the data integration domain, the input data set is static meaning that, at the data merging process, the size of the input data is known in order to construct the proper size tree. After the merging process, the tree is removed from the memory or temporary disk space.

Domain Spaces

As we mentioned early, the matching of the objects can be performed in either Euclidean space or Metric space. In the Euclidean space, the distance between two objects is measured by using the Euclidean function. In the Metric space, the distance between two objects is measured by using a function that satisfies symmetry, non-negativity, and triangular inequality properties. Each existing tree (e.g., M-tree, R-tree) was designed to suite for different object spaces. In the data integration domain, the tree should be optimized to this particular application domain; namely it should support object clustering in the Metric space, because we are matching the semistructured data objects (i.e., XML elements) that are correlated and whose schema is flexible.

Cluster Size

Cluster size determines the number of elements in a cluster. Should the size of each cluster be uniform (i.e., every cluster have the same number of elements)? Can the size of each cluster be variable (i.e., each cluster can have a different number of elements)? These and other questions need to be considered since the size of the cluster is parameter that indicates the number of errors as well as the difficulty of an algorithm for generating a clustering tree. If the size of each cluster is uniform, the algorithm for

generating a clustering tree is straightforward, since the number clusters in the tree is the ratio of the number of data items and the cluster size. If the size of each cluster is variable, the algorithm for generating a clustering tree is complicated since the number of clusters cannot be predicted. However, the variation of cluster size may be able to provide more accurate results because of a greater probability of placing data items that represent the same real-world entities into the same cluster and that represent the different real-world entities into different clusters.

Tree Operations

The most common operations on a tree are insertion and deletion. In the data integration domain, we need to consider trees that provide fast construction time i.e., the insertion of an element into the tree should be fast. Deletion of an element is not a major concern.

Design Approach

The tree can be built in either top-down or bottom-up fashion. In the top-down approach, as the first step, all elements are in one single cluster. The single cluster is decomposed into two or more sub-clusters based on the similarity of the object. Then, each sub-cluster is recursively decomposed until no cluster can be decomposed further (i.e., until the number of elements in each cluster is less than or equal to the maximum cluster size). In the bottom-up approach, as the first step, if there are *N* elements, there are *N* clusters each of which contains only one element. Next, clusters that are closed to each other (i.e., clusters of which distances between center points is below some threshold) are merged into bigger clusters. This process proceeds recursively until all elements are in a single cluster. In this research, we have considered both design approaches. Using the top-down approach, building a clustering tree can be expensive

because the comparisons between elements are repeated every iteration for decomposing a single cluster. Using the bottom-up approach, building a clustering tree can avoid repeating of comparison between elements since a new element can be located in a cluster as soon as it has been compared to an element in that cluster and the two elements are closed to each other (i.e., elements of which distances between them is below some threshold). However, the implementation of algorithm for building a clustering tree using the bottom-up approach is usually more complicated than that using the top-down approach.

Overlapping Clusters

A more complex situation arises when clusters overlap. Does the tree allow overlapping of clusters? Are elements restricted to membership in one cluster or can they exist in multiple clusters simultaneously? Those are some of the questions that we had to address. If we allow overlapping clusters, this situation may affect the computation time and the number of errors. Overlapping clusters can reduce false alarms (because they increase the probability that data items representing the same real-world object are in the same cluster). However, it can increase false dismissals because the same item can appear in different clusters; hence the overall error, which consists of both false dismissals and false alarms, may increase. Additional comparisons among elements in the overlapping areas may have to be performed to reduce error.

Split/Decomposition Strategies

The term "split" is normally used for trees that are designed in a bottom-up fashion whereas the term "decomposition" is normally used for trees that are designed in a top-down fashion. In the top-down approach, the decomposition strategy designates the shape (structure) of the trees. In the bottom-up approach, the split strategy must give the

split result that satisfies the tree properties (e.g., every node in the tree contains entries no more than 70% of the node capacity). The split/decomposition strategies are very important because they account for most of the construction time of the tree. Different strategies can have different goals. For example, the *ball decomposition* guarantees the balance of the tree for top-down construction. Most strategies used in the bottom-up construction attempt to achieve minimum-volume (size) and minimum-overlap of the clusters. As the first step of the decomposition, a focal object, called a *pivot object*, is selected. Some split strategy such as *generalized hyperplane* may need more than one pivot object. The pivot object(s) can be selected randomly from the objects in the same cluster. Second, the distance from the pivot object(s) to all other objects in the same cluster are calculated. Then, the cluster is decomposed based on some heuristic. For example, one heuristic is to randomly pick the distance that has been calculated in the second step. Another heuristic is to pick the median of the distances that has been calculated in the second step. More complicated heuristics can be applied as well.

The framework explained above will be used to investigate and evaluate heuristics for building our clustering tree. As we have mentioned, our hierarchical clustering model contains a clustering tree and a set of distance functions. In the next section, we will provide the detailed implementation of distance functions in our hierarchical clustering model.

Distance Functions

The original purpose of the metric-tree is to provide an index structure to speed up range and nearest neighbor queries. The implementation of the distance functions for the metric-trees is considered as a black box meaning that different sets of distance functions

can be implemented differently in different application domains. Choosing an appropriate distance functions is very important since it is used to measure the (dis)similarity between two elements. Distance functions are sensitive to their application domain and need to be chosen based on the types of comparisons made.

For those trees that operate in the Euclidean space, selecting a distance function is trivial since the Euclidean function itself can be applied. Instead, the problem is how to select an appropriate transformation function that transforms an object into a point in the Euclidean space. A good transformation function should preserve the original distance between two objects after they have been mapped into points in the Euclidean space.

For those trees that operate in the Metric space, the distance function must satisfy the symmetry, non-negativity and triangular inequality properties as described in Chapter 2. To satisfy the symmetry property, the distance from an object *x* to an object *y* must be equal to the distance from object *y* to object *x*. To satisfy the non-negativity property, the distance for each pair of different objects must be positive and the distance from an object to itself must be zero. To satisfy the triangular inequality property, the distance from an object *x* to an object *y* must not be greater than the sum of the distances from object *x* to another object *z* and the distance from object *z* to object *y*.

Since we use a metric-tree to match objects, finding appropriate distance functions that are well suited for the metric space is a major focus. The objects that we are dealing with can be primitive or complex. A primitive object is an object that contains a single value. A complex object is an object that contains one or more objects which can be either primitive or complex. Therefore, suitable distance functions can be classified into two categories: Primitive distance functions, and complex distance functions.

Primitive Distance Functions

The distance between two primitive objects can be defined based on the type of the data value. For binary data types (e.g., yes/no, female/male), the distance between two primitive objects may be defined as 0 or 1 depending on their values. For numeric data types (e.g., integer, float), the distance between two primitive objects may be defined as the absolute value of the difference of the their values. For nominal data types (e.g., color can be either blue, brown or black), the distance between two primitive objects must be obtained from the user based on the user's viewpoint for matching the values of the objects. For string data types, the distance between two primitive objects may be their edit distance which can obtained by counting the number of insertions, deletions, and changes of the characters that separate the two strings. The distance between two strings representing English terms can be defined based on synonym, homonym, and acronym relationships between the two strings. In addition, the distance between two strings can also be defined based on a typographical error.

For our research, since the data we are dealing with are semi-structured and are represented in XML format, the type of data values inside primitive objects are strings. Hence, we are using the edit distance function to measure the similarity of two semistructured objects. Other distance functions for other data types may be applied in the future. The edit distance can be obtained by finding the longest common subsequence of characters.

Complex Distance Functions

The distance between two complex object instances can be recursively derived from the distances between the corresponding sub-objects. Unlike structured data objects whose schema (i.e., structure) is rigid, semistructured data objects have flexible

structures. Therefore, the distance function for complex object instances must capture the features that compare the contents of such complex object instances and that can compare those instances whose contents are partially equivalent. In XML terminology, we refer to an object instance as element which can contain other elements and which can have a set of associated attributes. The object instances are declared in a document as element instances, whereas their structures can be defined in an accompanying DTD which may be stored in the same document or in a different file.

Let C_1 and C_2 be two element instances of a complex element C. Let S_i^p be the element instance of element S^p for some running numbers p and i. The element S_i^p forms the content of element C. Let A_i^q be the value of attribute A^q, for some running numbers q and j. The attribute A^q is associated with element C. We model the distance function between instances C_1 and C_2 based on a weighting scheme of element C. Let ${}_EW^p$ and ${}_AW^p$ be the weight values associated with element S^p and attribute A^p of element C, respectively. In general, the distance function, Δ, between the two instances of the complex element C is defined as follows:

$$Dist(C_1, C_2) = \Delta C = \sum_{p=1}^{n} ({}_EW^p \Delta S^p) + \sum_{q=1}^{m} ({}_AW^p * \Delta A^q)$$

If element S^p is a complex element, ΔS^p is defined in the same way as ΔC. Otherwise, the element S^p is a primitive element; thus, a primitive distance function is applied (e.g., an edit distance function is applied when the element S^p is type of string). The distance, ΔA^q, between two attribute values is defined as a primitive distance function for a particular attribute type (e.g., numeric, binary, string, qualitative).

In XML, there are four types of element constraints e.g., *zero-or-one, one-or-more, zero-or-more,* and *exactly-one,* and four types of attribute constraints e.g. *#IMPLIED, #REQUIRED, #FIXED* and *DEFAULT*. Because of variations of element and attribute constraints, the distance function for each type of constraint is provided as follows. Consider a complex element C that contains a list of elements one of which is element S^1. Element instances C_1 and C_2 are of type C. If the constraint associated with element S^1 is "exactly-one" (i.e., an instance of element C must contain exactly one instance of S^1), the distance between C_1 and C_2 can be computed as follows:

$$Dist(C_1, C_2) = \Delta C = ({}_E W^1 * \Delta S^1 + \ldots)$$

If the constraint associated with element S^1 is "zero-or-one", and the instances of S^1 are declared inside both C_1 and C_2, the distance between C_1 and C_2 is the same as in the case of the exactly-one constraint. However, if there is only one instance of S^1 which is declared inside one of the two element instances, say C_1, the distance between C_1 and C_2 can be computed as

$$Dist(C_1, C_2) = \Delta C = (q + {}_E W^p * \Delta S^p + \ldots),$$

where q is equal to 0, if ${}_E W^p * \Delta S^p \neq 0$, for $\exists p \in \mathrm{N}$ and $p \neq 1$. Otherwise, it is equal to infinity. We will use the notation `element1.element2` to denote the fact that `element2` is a child of `element1`. Let ${C_1}.^i S^1$ and ${C_2}.^i S^1$ be the i^{th} instances of S^1 declared in C_1 and C_2, respectively. If the constraint associated with element S^1 is "one-or-more", the distance between C_1 and C_2 is as follows:

$$Dist(C_1, C_2) = \Delta C = ({}_E W^1 * \min_{i,j}\{Dist({C_1}.^i S^1, {C_2}.^j S^1)\} + \ldots)$$

If the constraint associated with element S^1 is "zero-or-more", the distance between C_1 and C_2 is derived from the cases for one-or-more and zero-or-one constraints.

In case of attribute constraints, consider attribute A^1 of element C. Let A_1^1 and A_2^1 be attributes declared inside C_1 and C_2, respectively. Let V_1^1 and V_2^1 be values of the two attributes. Let ${}_AW^1$ be the weight of attribute A^1. If the constraint of A^1 is "#REQUIRED", the distance between C_1 and C_2 is as follows:

$$Dist(C_1, C_2) = {}_AW^1 * Dist(V_1^1, V_2^2) + \ldots.$$

Note that the distance function for attribute values is primitive. If the constraint of A^1 is "#IMPLIED", and if one of the two attribute values is null, the distance between C_1 and C_2 is as follows:

$$Dist(C_1, C_2) = \Delta C = (q + {}_AW^q * \Delta A^q + \ldots),$$

where q is equal to 0, if ${}_AW^q * \Delta A^q \neq 0$, for $\exists q \in \mathrm{N}$ and $q \neq 1$. Otherwise, it is equal to infinity. If the constraint of A^1 is "#FIXED", V_1^1 and V_2^1 must be fixed and both are identical; hence, the distance between the two values is always zero. If the constraint of A^1 is DEFAULT, and if one of the two values is not declared, the default value will be used; hence, the distance is obtained by comparing the default value with the other.

To compare object instances in the Metric space, it is required that the distance functions that we use must satisfy the symmetry, non-negativity, and triangular properties. Since we are using the edit distance function, the argument that this distance function satisfies the first two properties is straightforward. We currently use the edit distance function for comparing primitive objects. The distance between two string values is always non-negative. In addition, the distance from the first string to the second string is equal to the distance from the second string to the first string. The edit distance

function also preserves the triangular property. Our distance function for complex elements is derived from primitive distance functions. It contains only the weighted summation and multiplication of primitive distances. Since underlying primitive function never produces negative values, the non-negativity property is satisfied. Moreover, since the primitive function preserves the triangular property, so does the function for comparing complex elements. The distance from one complex element instance to another is equivalent to the reverse distance. Thus, the symmetry property is also preserved.

One component in our clustering model is a set of distance functions that can be either a primitive or a complex distance function. Another component in our clustering model is a clustering tree for which a framework has been explained in the first section of this chapter. Choosing an appropriate clustering tree to conform data items depends on a heuristic that is used to build the clustering tree. In the next section, we will introduce a set of heuristics that we have investigated, evaluated and implemented.

Clustering Heuristics

In this research project, we provide three heuristics for building the clustering tree. They are *All-Pair-Comparisons-based heuristic* (AC), *Selected-Comparison-based heuristic* (SC), and *M-tree-based heuristic* (MT). The algorithms for building the clustering tree using these heuristics are provided in this section. The qualitative analysis of those heuristics is deferred to the next chapter.

All-Pair-Comparisons-Based Heuristic (AC)

When this heuristic is used to build the clustering tree, the distances for every pair of items are calculated. Given a set of N items and the maximum number k of items that can be placed into the same cluster. The clustering tree is constructed as follows:

1. For each pair of items, calculate the distance between them. Let D be the list of all distances. Since the distance from item i to item j is equal to the distance from item j to item i, the size of the list is $N(N+1)/2$.

2. Sort the distances in D. Let D^* be the sorted list. Each entry of D^* contains the distance between two items.

3. Based on the sorted list D^* and the maximum cluster size k, generate the clustering tree containing $\lceil N/k \rceil$ clusters. Let A be a list of size N. The j^{th} entry of A indicates the cluster to which item j belongs. Initially, each entry of A is null (i.e., no cluster is assigned for each item). For each entry in D^*, extract the identification number of items i, and j. By starting from cluster 1, if the current cluster is not full, and both items i and j are not assigned into a cluster, place them into the current cluster. Otherwise, place them into the next cluster. Update the assignment list.

Theoretically, the run-time complexity of this algorithm is $O(N^2 logN)$ time; the space complexity is $O(N^2)$. The time taken in Step 2 dominates the time taken in other steps. The space for the distance list D indicates the overall space used in the algorithm.

Selected-Comparisons-Based Heuristic (SC)

Unlike the AC heuristic, SC randomly picks a set of items, called *pivots*. Then, it compares all other items to these pivots. Each pivot is the representative of each cluster that will be generated. Using the SC heuristic, the clustering tree is created as follows:

1. Given N items and maximum cluster size k, pick $\lfloor N/k \rfloor$ pivots from the item set. Calculate the distance from each pivot to all other items in the set. Let D be a matrix of size $N \times \lfloor N/k \rfloor$ that stores the distances between items and the pivots. Each row of matrix D represents an item. Each column of the matrix D represents a pivot.

2. Sort the distances for each column of matrix D; then, sort the distances for each row. Let D^* be the sorted matrix.

3. Allocate the clustering tree containing $\lceil N/k \rceil$ clusters. Each cluster will contain exactly k items except the last cluster that will contain (N mod k) items.

4. For each row i of matrix D^*, look for column j such that $D^*[i,j]$ is a minimum. Let $[p,q]$ be the original position of $D^*[i,j]$ in matrix D. If the cluster q is not full, assign item p to cluster q. Otherwise, find the next minimum value in row i of matrix D*, whose original position in matrix D is $[p,r]$ and cluster r is not full. Assign item p to cluster r. For every item p that is assigned to cluster q, record the assignment into an assignment list, A, of size N. The p^{th} entry of A indicates the cluster which item p belongs to. Initially, each entry of A is null (i.e., no cluster is assigned to an item).

5. For each entry i in A, if $A[i]$ is null, assign item i to the last cluster (i.e., the $\lceil N/k \rceil^{th}$ cluster.)

Theoretically, the run-time complexity of the algorithm is $O([N^2 logN]/k)$ time; the space complexity is $O(N^2/k)$. The time taken in Step 2 dominates the time taken in other steps. The space for the matrix D indicates the overall space used in the algorithm. Note that the algorithm that uses the SC heuristic is faster and uses less space with the factor of k than the AC-heuristic based algorithm.

M-tree-Based Heuristic (MT)

Originally, the M-tree [13] was introduced as an index structure for large spatial data sets to speed up data comparisons in the Metric space. As an index structure, the M-tree is used to support range and k-nearest-neighbor queries and to speed up the response time for searching and querying the data in the set. We are using the M-tree as a clustering tree to reconcile data. The tree should be built fast and must fit into memory.

An M-tree consists of a set of nodes each of which contains a list of entries and a pointer to its parent. Each entry of an internal node of M-tree stores a *routing-object* that is used for searching a place in the tree to insert a new object (note that we are only focusing on insertion of the new objects). In addition, the entry of the internal node contains a pointer to the root of the sub-tree, covering radius of the routing-object and the distance of the routing-object to its parent. The covering radius is the maximum distance of the distances from the routing-object to its children. Each entry in a leaf node of an M-tree stores an object or a pointer to the object and the distance from the object to the (routing-) object stored in the parent node.

The M-tree can be built by inserting the first object into an empty tree and continuing to insert new objects into a node of the tree, starting from the root and using the routing-object to obtain the path to the target node where the new object should be stored. During the insertion, if the target node is a full leaf node, the node overflows and it must be split. Splitting the overflow node creates a new node at the same level of the overflow node. Some entries inside the overflow node are transferred to the new node with using a partitioning criterion; hence, the overflow node becomes a non-empty node. After that, the new node is promoted to its parent. If the parent node is full, it must be split as well. If the source node that is supposed to be split is the root, a new root node is allocated and stores into the new root the source node and the other node obtained from splitting.

The performance of the M-tree with respect to indexing or clustering data objects depends on the policy that is used for splitting the tree. Ideally, the split policy should promote the routing-objects and partition other objects so that a routing-object has a small covering radius and has small overlap with other routing-objects. Ciacia et al. [13] introduce several strategies for picking routing objects. Our M-tree uses random strategy to pick two routing objects from the set of object in the overflow node. This strategy is very fast, simple and easy to implement. For the partitioning objects from the overflow node, the experimental made by Ciacia et al. [13] showed that the *generalized hyperplane decomposition* performs very well with the random strategy for splitting the overflow nodes; thus, we decided to use the generalized hyperplane in connection with our M-tree. Recall that, using the generalized hyperplane decomposition, objects are conceptually partitioned into three sets based on two pivots. The first set contains objects each of

which is closer to the first pivot. The second set contains objects each of which is closer to the second pivot. The third set contains the remains. However, since splitting an M-tree node requires two output nodes, we combine together the second and the third sets of the partition generated by the generalized hyperplane decomposition

Based on the framework for choosing a clustering tree that we described in the first section of this chapter, we investigated clustering strategies that exist in data mining and spatial access methods, such as M-tree [13], and came up with two new heuristics, which are the All-pair-comparison-based heuristic and Selected-comparison-based heuristic. We have developed the three heuristics which are appropriate for constructing a clustering tree in the data integration domain where the data items are in a multi-dimensional space and may be highly correlated. Note that the original M-tree is implemented using C++; we adopted the algorithm of constructing an M-tree and implemented it using Java. Clustering trees that are constructed by each of the three heuristics do not contain an overlapping cluster. The implementation of the three heuristics assumes that the clustering tree and all data items in a given data set must be fit into memory. Two data items are compared using both primitive and complex distance functions depending on the structure of the data items. The distance between two complex data items involves weight values that are associated with sub-structure of those data items. The weight values are one of parameters that are used during the clustering process, and defined within a complex distance function. The distance functions we use suit for comparing data items in Metric space where the distance functions must satisfy the symmetry, non-negativity, and triangular inequality properties, as described in Chapter 2. Using the All-pair-comparison-based and Selected-comparison-based

heuristics, a clustering tree is built in a top-down fashion because, as the first step, all elements are in one single cluster. Using the M-tree based heuristic, a clustering tree is built in a bottom-up fashion because, at the beginning, each element is in an individual cluster.

Our hierarchical clustering model is actually part of an overall framework for resolving conflicts as described in our technical report [56]. It is introduced to resolve data conflicts such as synonyms, homonyms, acronyms, ID-values, missing data, and spelling mistakes. In our integration system, the wrappers resolve structural and domain conflicts such as aggregation, generalization, scale/unit and precision conflicts. After that, the mediator compares the similarity among elements using both primitive and complex distance functions. Then, the clustering tree is built. At the end of building the tree, all objects from two or more sources are in one of the clusters that are at the leaf level of the tree. Next, further matching among the elements in a leaf cluster may proceed. If the matching continues, the shortest distance of each pair of elements indicates the similarity. Otherwise, we assume all objects in the same cluster are matched.

CHAPTER 7
QUALITATIVE ANALYSIS OF THE OBJECT MATCH TECHNIQUES

In this chapter, we provide experimental results on the performance of the three heuristics mentioned in the previous chapter. The experiments evaluate the time complexity of the algorithms for constructing a clustering tree and the accuracy of the result of the element matching using the clustering tree. The accuracy is obtained by counting the number of false alarms and false dismissals. Based on our framework mentioned in the previous chapter, the experiments measure not only the efficiency of the algorithms but also the size of cluster (i.e., the maximum number of elements that can be in the same cluster) and the size of input data (i.e., the number of input elements.)

Description of the Test Data

The data used in our experiments was modified from a sample bibliography data set that was enclosed in the package of the XML-QL processor [3] developed at the AT&T Research Lab. The original bibliography data set contains 722 journal articles, 247 books, 6 book chapters, 100 collections, 635 proceedings articles, 23 manuals, 2 Master's theses, 40 Ph.D.'s dissertations, and 145 technical reports. This data set is used as one of the data sources for our integration system prototype. Each publication is represented as an XML element. Note that the data set contains no duplicate elements. Since the purpose of the experiments is to evaluate the efficiency of our hierarchical clustering model for data reconciliation (e.g., matching the elements and remove all the duplicates), we made the following changes: We chose two subsets of the bibliography

data set and generated duplicates including some variations of the data values (e.g., adding extra characters for each #PCDATA element) to simulate the fact that data values representing the same real-world concept from different sources may not match exactly. The first data set described above contains 20 Ph.D. dissertations.

In addition, we changed the structure of Ph.D. dissertation elements to make it consistent with the structure of Ph.D. dissertation elements defined in the global Ontology DTD (see Appendix A for the full definition) that is used in our IWIZ integration system. In IWIZ, this task is conducted by the wrappers which extract data from sources and convert them into the IWIZ internal data model of application schema. The structure of Ph.D. dissertation elements is shown in Figure 9, and contains one of `Author`, `Title`, `School`, and `Year`, respectively, as well as zero-or-one of `Address`, `Month`, `Type` and `Note`, respectively. Each `Author` contains one `Lastname` as well as an optional `Firstname` and an optional `Address`.

```
<!ELEMENT PhdThesis (Author, Title,
                     School, Address?,
                     Year,   Month?,
                     Type?,  Note?  )>
<!ELEMENT Address (#PCDATA)>
<!ELEMENT Author (Firstname?, Lastname, Address?)>
<!ELEMENT Firstname (#PCDATA)>
<!ELEMENT Lastname (#PCDATA)>
<!ELEMENT Month (#PCDATA) >
<!ELEMENT Note (#PCDATA) >
<!ELEMENT School (#PCDATA) >
<!ELEMENT Title (#PCDATA) >
<!ELEMENT Type (#PCDATA) >
<!ELEMENT Year (#PCDATA) >
```

Figure 9: Sample DTD describing the structure of the concept “Ph.D. Thesis”

Our second data set is also a bibliography data set and contains 20 articles. The structure of an `Article` element is shown in Figure 10. Each `Article` element

contains one or more elements of type `Author`, one element each of type `Title`, `Year` and `Journal`, and zero-or-one element of type `Address`, `Month`, `Type` and `Note`. Each `Author` contains one `Lastname` and optional `Firstname` and `Address`. Each `Journal` contains one `Title`, and optional `Year`, `Month`, `Volume` and `Number`.

```
<!ELEMENT Article ( Author+, Title, Year, Month?,
                    Pages?, Note?, Journal )>
<!ELEMENT Address (#PCDATA)>
<!ELEMENT Author (Firstname?, Lastname, Address?)>
<!ELEMENT Firstname (#PCDATA)>
<!ELEMENT Lastname (#PCDATA)>
<!ELEMENT Journal (Title, Year?, Month?,
                   Volume?, Number? )>
<!ELEMENT Month (#PCDATA) >
<!ELEMENT Note (#PCDATA) >
<!ELEMENT Number (#PCDATA) >
<!ELEMENT Title (#PCDATA) >
<!ELEMENT Type (#PCDATA) >
<!ELEMENT Volume (#PCDATA) >
<!ELEMENT Year (#PCDATA) >
```

Figure 10: Sample DTD describing the structure of the concept “Article”

After creating the two data sets containing elements whose structure corresponds to the ontology, we first assigned a unique identifier to each element instances in each data set. The identifier number will help us to measure the accuracy of our clustering model.

Using the sample base data sets, we synthesized six different data, which contain 50, 100, 150, 200, 300, and 400 data elements, respectively. In each data source, each element is duplicated by 5% of the total size (e.g., in data set of size 100, there are 5 copies of each data element). For each duplicate, we add two variations of the data values (i.e., two extra characters for each PCDATA values.) For each synthetic data set, we calculate the minimum upper bound percentage of errors, as shown in Column 4 of Table 1. That percentage indicates the number of matching errors relative to the total

number of data elements with no removal. That percentage number is used as a basis for our evaluation.

Table 1: Characteristics of data sets

Data set	Size of data set (items)	Average number of duplicates per element (items)	Base error (in %)
PhdThesis	50	2.5	4.00
PhdThesis	100	5.0	4.79
PhdThesis	150	7.5	4.76
PhdThesis	200	10.0	4.86
PhdThesis	300	15.0	4.98
PhdThesis	400	20.0	4.96
Article	50	2.5	4.16
Article	100	5.0	4.67
Article	150	7.5	4.85
Article	200	10.0	4.93
Article	300	15.0	4.94
Article	400	20.0	4.90

Experimental Results and Discussion

In this section, we provide experimental results on performance and accuracy of our hierarchical clustering model. We tested all three heuristics – All-Pair-Comparison-based heuristic, Selected-Comparison-based heuristic, and M-tree-based heuristics – that are described in the previous chapter. The experiments were based on the synthetic data sets described in the previous section. The complete results of the following experiments are summarized in Appendix B. In this chapter, we summarize our analysis of the experiments, focusing on several key experiments that illustrate the characteristics of the different clustering techniques.

Table 2: Results on "PhdThesis" data sets for all-pair-comparison-based (AC), selected-comparison-based (SC), and M-tree-base (MT) heuristics with uniform weights using terms that are separated by no more than two variations in the spelling

Test Description	Time (sec)			False alarm (%)			False dismissal (%)			Error (%)		
	AC	SC	MT	AC	SC	MT	AC	SC	MT	AC	SC	MT
50 items 3% cluster size	4	4	3	0.24	0.76	0.63	2.29	2.71	3.45	2.53	3.47	4.08
50 items 5% cluster size	4	2	2	0.33	2.31	1.49	2.29	2.31	3.12	2.61	4.61	4.61
50 items 7% cluster size	4	2	2	4.24	3.88	2.12	2.29	1.92	2.49	6.53	5.80	4.61
50 items 10% cluster size	4	1	2	4.24	6.35	2.98	2.29	2.18	2.18	6.53	8.53	5.16
50 items 15% cluster size	4	1	2	12.08	11.55	6.76	2.29	1.76	1.57	14.37	13.31	8.33
50 items 20% cluster size	4	1	1	15.92	15.73	9.02	2.29	1.37	1.51	18.20	17.10	10.53
100 items 2% cluster size	14	14	6	0.14	0.25	0.34	3.92	4.03	4.52	4.06	4.27	4.86
100 items 3% cluster size	14	9	5	0.14	0.63	0.59	3.92	3.41	4.11	4.06	4.04	4.70
100 items 5% cluster size	14	5	4	1.76	1.93	1.15	3.52	2.68	3.31	5.27	4.62	4.46
100 items 7% cluster size	14	4	3	5.64	4.35	2.35	3.52	2.23	2.53	9.15	6.59	4.88
100 items 10% cluster size	14	3	3	7.58	6.56	3.58	3.27	2.25	2.01	10.85	8.81	5.59
100 items 15% cluster size	14	2	3	11.29	11.20	6.13	3.19	2.35	1.73	14.48	13.55	7.85
100 items 20% cluster size	14	1	2	17.56	16.47	10.42	3.15	2.07	1.39	20.71	18.55	11.81
150 items 2% cluster size	34	22	8	0.04	0.32	0.31	4.13	3.74	4.24	4.18	4.07	4.55
150 items 3% cluster size	34	14	8	1.33	0.84	0.81	4.10	2.91	3.79	5.43	3.75	4.61
150 items 5% cluster size	34	8	6	3.91	2.21	1.15	4.03	2.33	2.81	7.94	4.55	3.96
150 items 7% cluster size	34	6	6	5.27	3.75	2.02	3.99	1.93	2.06	9.26	5.68	4.09
150 items 10% cluster size	34	4	6	7.63	6.68	3.27	3.85	2.04	1.58	11.48	8.72	4.85
150 items 15% cluster size	34	3	4	12.55	11.59	6.81	3.54	2.17	1.43	16.10	13.76	8.25
150 items 20% cluster size	34	2	3	17.54	16.36	9.92	3.10	1.66	1.33	20.64	18.02	11.25
200 items 2% cluster size	62	30	11	0.88	0.28	0.32	4.25	3.63	4.22	5.14	3.91	4.54
200 items 3% cluster size	62	21	10	1.84	0.76	0.52	4.21	3.13	3.73	6.06	3.89	4.25
200 items 5% cluster size	62	13	9	3.85	1.76	1.16	4.23	2.10	2.81	8.08	3.86	3.97
200 items 7% cluster size	62	8	8	5.62	4.73	2.12	4.05	2.68	2.25	9.67	7.42	4.37
200 items 10% cluster size	62	6	8	8.52	6.64	3.42	3.93	1.95	1.65	12.45	8.60	5.07
200 items 15% cluster size	63	4	6	12.92	11.63	7.30	3.71	2.42	1.54	16.63	14.05	8.84
200 items 20% cluster size	63	3	5	17.94	16.04	9.90	3.40	1.31	1.13	21.34	17.35	11.03
300 items 2% cluster size	123	39	16	1.13	0.36	0.25	4.44	3.66	4.13	5.58	4.02	4.38
300 items 3% cluster size	124	28	14	1.64	0.68	0.58	4.30	3.00	3.69	5.94	3.68	4.27
300 items 5% cluster size	123	16	13	3.52	2.08	1.04	4.20	2.37	2.68	7.72	4.45	3.72
300 items 7% cluster size	121	11	13	5.87	4.22	2.14	3.95	2.30	2.50	9.81	6.52	4.65
300 items 10% cluster size	121	8	10	8.53	6.73	3.79	3.81	2.01	1.91	12.35	8.75	5.71
300 items 15% cluster size	122	6	8	12.54	11.41	6.94	3.46	2.18	2.03	16.00	13.59	8.97
300 items 20% cluster size	121	4	7	17.92	16.78	9.60	3.17	2.03	1.87	21.09	18.80	11.46
400 items 2% cluster size	233	57	23	1.24	0.40	0.30	4.45	3.61	4.09	5.69	4.01	4.39
400 items 3% cluster size	232	39	22	2.14	0.83	0.51	4.36	3.05	3.60	6.50	3.88	4.11
400 items 5% cluster size	232	23	20	4.04	1.77	1.25	4.27	1.97	2.66	8.31	3.73	3.91
400 items 7% cluster size	232	17	17	5.86	4.45	2.21	4.15	2.48	1.99	10.01	6.93	4.20
400 items 10% cluster size	232	12	14	8.71	6.71	3.46	3.95	1.90	1.58	12.66	8.61	5.04
400 items 15% cluster size	233	8	11	12.90	11.32	6.97	3.58	2.00	1.93	16.48	13.32	8.90
400 items 20% cluster size	234	5	9	18.07	16.49	10.71	3.33	1.65	1.15	21.40	18.14	11.86

Table 3: Results on "Article" data sets for all-pair-comparison-based (AC), selected-comparison-based (SC), and M-tree-base (MT) heuristics with uniform weights using terms that are separated by no more than two variations in the spelling

Test Description	Time (sec)			False alarm (%)			False dismissal (%)			Error (%)		
	AC	SC	MT	AC	SC	MT	AC	SC	MT	AC	SC	MT
50 items 3% cluster size	7	7	5	0.65	0.98	0.76	2.78	3.10	3.67	3.43	4.08	4.43
50 items 5% cluster size	7	5	4	0.65	2.45	1.76	2.78	2.61	3.45	3.43	5.06	5.20
50 items 7% cluster size	7	3	3	4.57	4.29	2.69	2.78	2.49	3.08	7.35	6.78	5.78
50 items 10% cluster size	7	3	3	4.57	6.27	3.78	2.78	2.27	2.80	7.35	8.53	6.57
50 items 15% cluster size	7	2	2	12.41	11.73	9.14	2.78	2.10	1.92	15.18	13.84	11.06
50 items 20% cluster size	7	1	2	16.49	16.51	9.47	2.29	2.31	2.00	18.78	18.82	11.47
100 items 2% cluster size	27	27	10	0.16	0.39	0.37	3.82	4.05	4.41	3.98	4.44	4.78
100 items 3% cluster size	27	18	8	0.16	0.97	0.68	3.82	3.64	4.01	3.98	4.61	4.68
100 items 5% cluster size	27	10	7	2.06	2.53	1.55	3.70	3.16	3.62	5.76	5.69	5.16
100 items 7% cluster size	27	7	5	5.82	5.25	3.38	3.58	3.01	2.93	9.39	8.25	6.32
100 items 10% cluster size	27	5	6	7.84	7.02	4.03	3.41	2.60	2.51	11.25	9.62	6.53
100 items 15% cluster size	27	4	5	11.66	11.74	7.55	3.43	2.77	2.09	15.09	14.52	9.64
100 items 20% cluster size	27	2	4	17.66	17.41	11.13	3.13	2.89	2.06	20.79	20.30	13.19
150 items 2% cluster size	63	42	15	0.11	0.50	0.41	4.29	4.01	4.42	4.39	4.51	4.82
150 items 3% cluster size	64	24	13	1.32	1.05	0.78	4.17	3.22	3.86	5.49	4.27	4.64
150 items 5% cluster size	64	16	13	3.79	2.42	1.51	3.99	2.63	3.07	7.78	5.05	4.58
150 items 7% cluster size	64	11	10	5.25	4.00	2.65	4.06	2.26	2.85	9.32	6.26	5.50
150 items 10% cluster size	63	8	10	7.49	7.03	4.48	3.79	2.49	2.75	11.28	9.52	7.23
150 items 15% cluster size	63	6	7	12.55	11.68	7.38	3.62	2.36	2.05	16.17	14.04	9.43
150 items 20% cluster size	63	4	7	18.19	16.85	10.81	3.58	2.23	2.20	21.77	19.08	13.01
200 items 2% cluster size	109	53	17	0.89	0.49	0.44	4.33	3.92	4.37	5.22	4.42	4.81
200 items 3% cluster size	109	35	16	1.90	0.86	0.70	4.34	3.30	3.89	6.24	4.16	4.59
200 items 5% cluster size	109	21	15	3.80	2.16	1.49	4.26	2.57	3.23	8.07	4.72	4.72
200 items 7% cluster size	109	15	14	5.60	4.76	2.59	4.11	2.78	2.74	9.71	7.54	5.33
200 items 10% cluster size	109	10	14	8.42	6.79	3.84	3.90	2.17	2.24	12.32	8.96	6.08
200 items 15% cluster size	108	7	9	12.89	11.36	8.40	3.75	2.22	2.97	16.64	13.58	11.38
200 items 20% cluster size	109	5	8	17.83	16.75	10.89	3.36	2.08	2.14	21.19	18.83	13.04
300 items 2% cluster size	256	84	28	1.13	0.55	0.38	4.40	3.81	4.20	5.53	4.36	4.58
300 items 3% cluster size	257	57	26	1.71	0.93	0.67	4.31	3.21	3.76	6.02	4.14	4.43
300 items 5% cluster size	257	31	25	3.53	2.36	1.64	4.15	2.61	3.29	7.68	4.97	4.92
300 items 7% cluster size	257	22	23	5.87	4.37	2.54	3.91	2.41	2.69	9.77	6.79	5.23
300 items 10% cluster size	258	16	20	8.50	7.33	4.66	3.80	2.57	2.45	12.30	9.90	7.10
300 items 15% cluster size	259	11	14	12.58	12.03	7.81	3.46	2.75	2.20	16.04	14.78	10.02
300 items 20% cluster size	259	7	12	17.86	17.00	12.47	3.20	2.20	2.79	21.06	19.20	15.26
400 items 2% cluster size	437	108	36	1.22	0.78	0.39	4.38	3.93	4.09	5.60	4.70	4.48
400 items 3% cluster size	439	70	34	2.10	1.53	0.67	4.28	3.70	3.68	6.38	5.22	4.34
400 items 5% cluster size	437	42	31	3.84	3.55	2.25	4.01	3.69	3.70	7.86	7.24	5.95
400 items 7% cluster size	435	28	33	5.77	5.55	2.73	3.98	3.52	2.84	9.76	9.07	5.58
400 items 10% cluster size	436	21	24	8.39	8.35	5.36	3.57	3.48	3.31	11.97	11.83	8.67
400 items 15% cluster size	435	14	18	12.86	12.74	9.13	3.48	3.36	3.21	16.33	16.10	12.34
400 items 20% cluster size	439	9	16	18.03	17.88	13.17	3.23	2.99	3.05	21.26	20.87	16.22

There are two parts to the analysis of the experiments. The first part concentrates on time complexity. The second part concentrates on accuracy of the heuristics as it relates to the size of the data set.

Table 2 shows the results of testing the All-Pair-Comparison-based (AC), Selected-Comparison-based (SC), M-tree-based (MT) heuristics on various "PhdThesis" data sets with uniform weights, using duplicate terms that are separated by no more than two variations in spelling. Table 3 shows the results of testing the All-Pair-Comparison-based (AC), Selected-Comparison-based (SC), M-tree-based (MT) heuristics on various "Article" data sets with uniform weights, using terms that are separated by no more than two variations in spelling. The size of the data sets varies from 50 to 400 elements. The maximum cluster size varies from 2% to 20% of the total data set. We did not perform tests on data sets of size 50 with cluster size of 2% since each cluster would contain only one element and duplicates are not removed; hence, the query result is immediately returned without removing the duplicates.

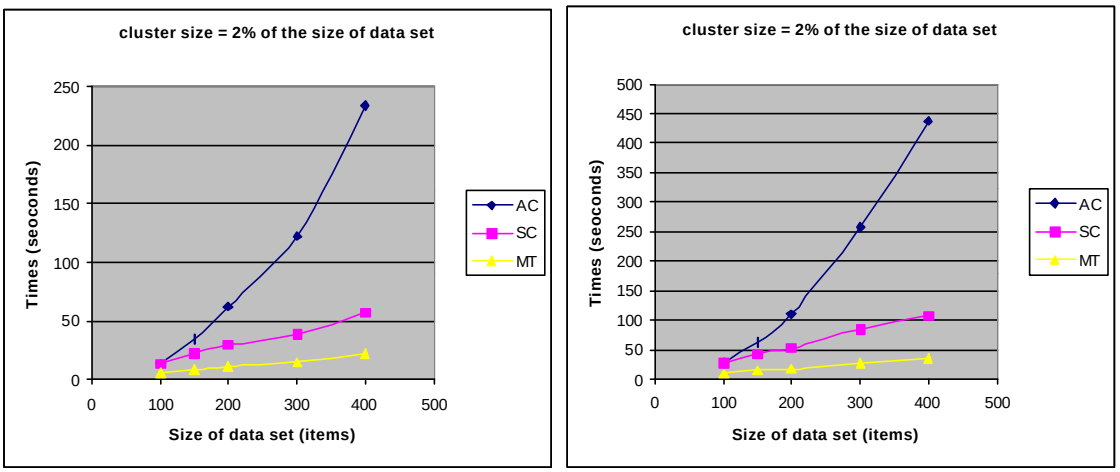

Figure 11: Time complexity of each clustering heuristic using 2% of the total data set as cluster size on "PhdThesis" (left) and "Article" (right) data sets

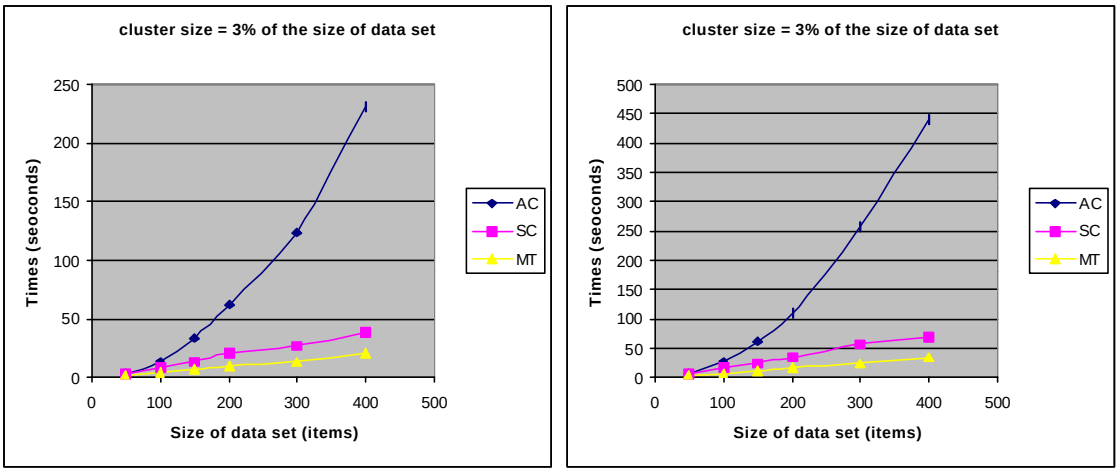

Figure 12: Time complexity of each clustering heuristic using 3% of the total data set as cluster size on "PhdThesis" (left) and "Article" (right) data sets

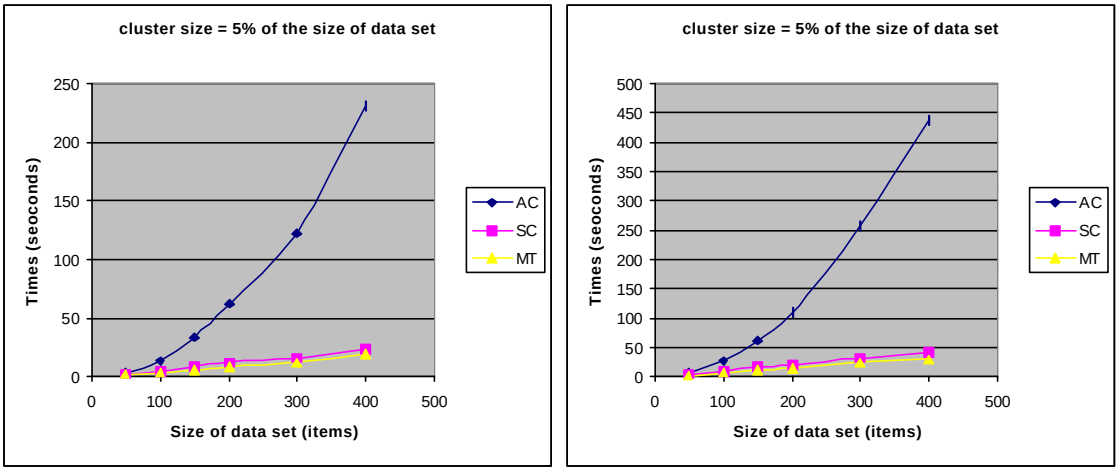

Figure 13: Time complexity of each clustering heuristic using 5% of the total data set as cluster size on "PhdThesis" (left) and "Article" (right) data sets

Let us consider the time complexity for each heuristic separately. Figure 11 through 13 show the time complexity of all three heuristics using cluster sizes of 2%, 3% and 5% of the total data set. The results on both "PhdThesis" and "Article" data sets exhibit the same trend although more time is spent on the "Article" data sets due to the more complex structure and higher number of data values in the data sets. In each case,

the MT heuristic performs fastest followed by the SC and the AC heuristics. For small data set (i.e., data sets of size 50 items), Figure 14 zooms in on time complexity for each heuristic while varying the cluster size.

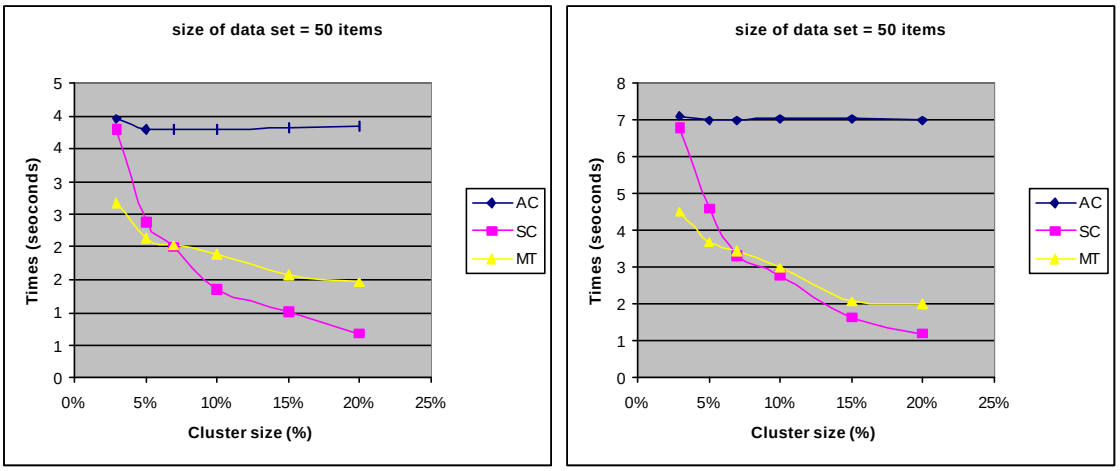

Figure 14: Time complexity for each heuristic on data set of size 50 items with variation in the cluster size on "PhdThesis" (left) and "Article" (right) data sets

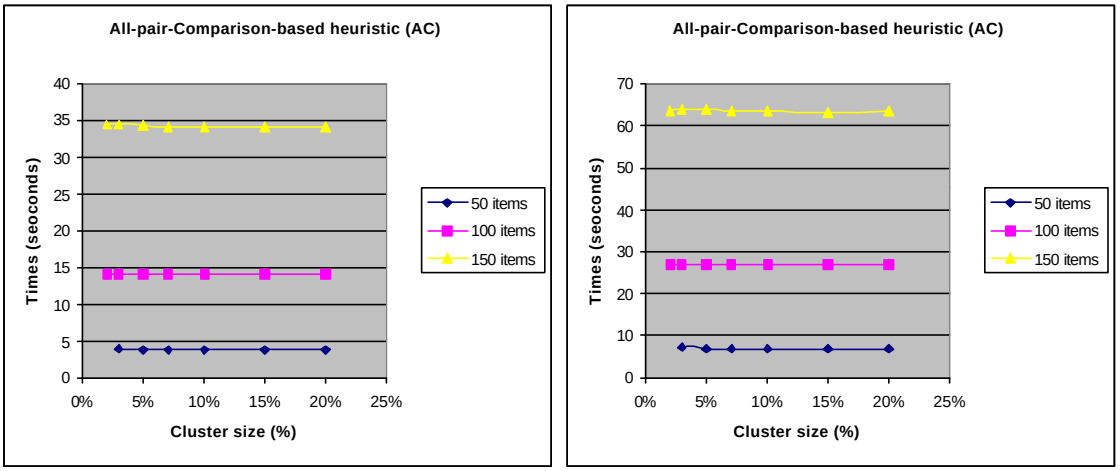

Figure 15: Time complexity for the All-pair-Comparison-based heuristic on small data sets with variation in the cluster size for "PhdThesis" (left) and "Article" (right) data sets

Figure 15 shows the time complexity for the AC heuristic on small data sets with variation in the cluster size. The time for constructing a clustering tree mainly depends

on the number of element comparisons. By using the AC heuristic, all elements are compared no matter how large the cluster size is. As we can see from Figure 15, the time it takes to construct a clustering tree using the AC heuristic is constant for the same data set and is not affected by the cluster size.

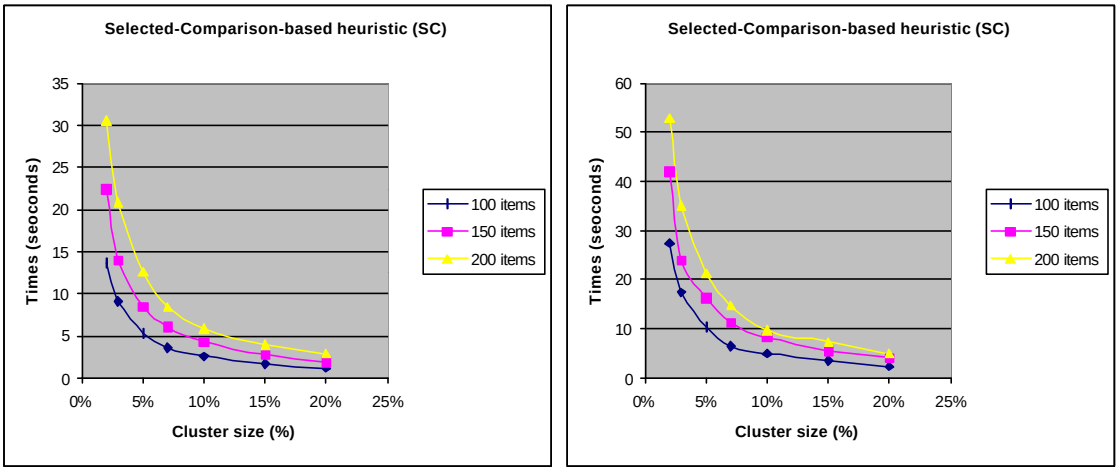

Figure 16: Time complexity for the Selected-Comparison-based heuristic on data sets of medium size with variation in the cluster size for "PhdThesis" (left) and "Article" (right) data sets

Figure 16 shows the time complexity for the SC heuristic on data sets of medium size with variation in the cluster size. We can see that the SC heuristic is very sensitive to the cluster size. In other words, the larger cluster size, the less amount of time it takes to construct the clustering tree using the SC heuristic. The time for constructing a clustering tree mainly depends on the number of element comparisons. By using the SC heuristic, the number of comparisons depends on the number of pivots, and the number of pivots is derived from the ratio of the size of data set to the cluster size. As a result, for a data set of fixed size, the time it takes to construct a clustering tree using SC heuristic is related to the cluster size.

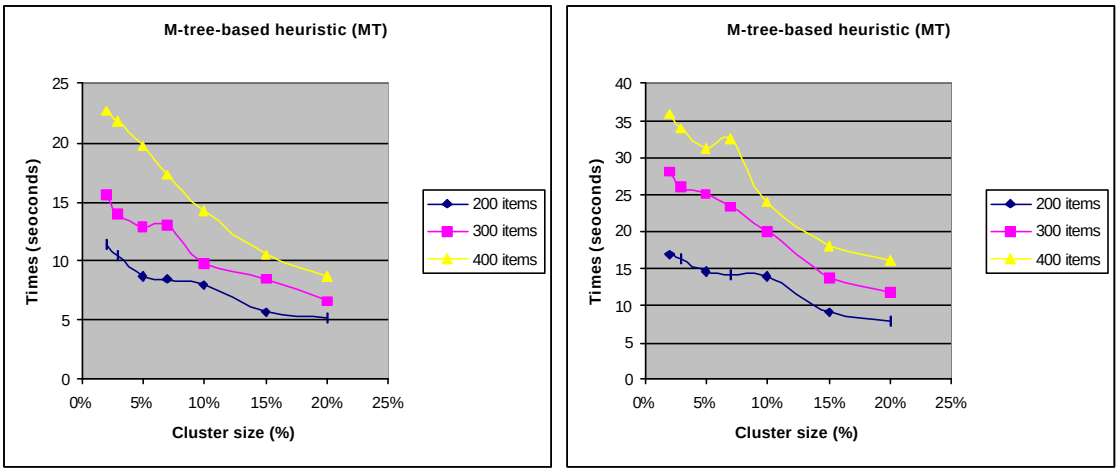

Figure 17: Time complexity for the M-tree-based heuristic on large data sets with variations in the cluster size for "PhdThesis" (left) and "Article" (right) data sets

Figure 17 shows the time complexity for the MT heuristic on large data sets with variation in the cluster size. As we expect, the larger the cluster size, the faster the element matching. This is because when the cluster size increases, the number of comparisons of elements decreases. The time complexity for each heuristic with variation of cluster size is shown in Figure 14. The figure only shows the result for data sets of size 50. We can see that the SC heuristic performs faster than the MT heuristic on the large cluster sizes.

The time complexities for the MT, SC and AC are $O(NlogN)$, $O([N^2logN]/k)$ and $O(N^2logN)$ respectively, where N is the size of the data set and k is the number of clusters. The time it takes to construct a clustering tree using the SC heuristic is obviously related to the cluster size as supported by our experimental result shown in Figure 16. In addition, our experimental result in Figure 15 supports the fact that the time it takes to construct a clustering tree using the AC heuristic is unrelated to the cluster size. Although, in theory, the time of constructing a clustering using MT heuristic does

not show that it is related to the cluster size, our experimental result in Figure 17 indicates the contrary. This is because the cluster size affects the maximum number of entries in an M-tree node. When inserting a new entry into a node, if the size of the node is large, the possibility that the node is full is small; hence the total time it takes to split a full node is reduced.

In theory, we expect that the SC should perform faster than the AC by a factor of *k*. However, according to the experiments in Table 2 and 3 we see that the SC executes run *k/2* times faster than the AC. For example, on the "PhdThesis" data set, if the size of data set is 100 items with the cluster size 3% of the total size, the absolute value of the cluster size is 3. In this case, SC takes 9 seconds while AC takes 14 seconds. This is because both heuristics spend most of their time on comparing elements. The number of comparison for the SC heuristic is kN^2 whereas that for the AC heuristic is $N(N+1)/2$.

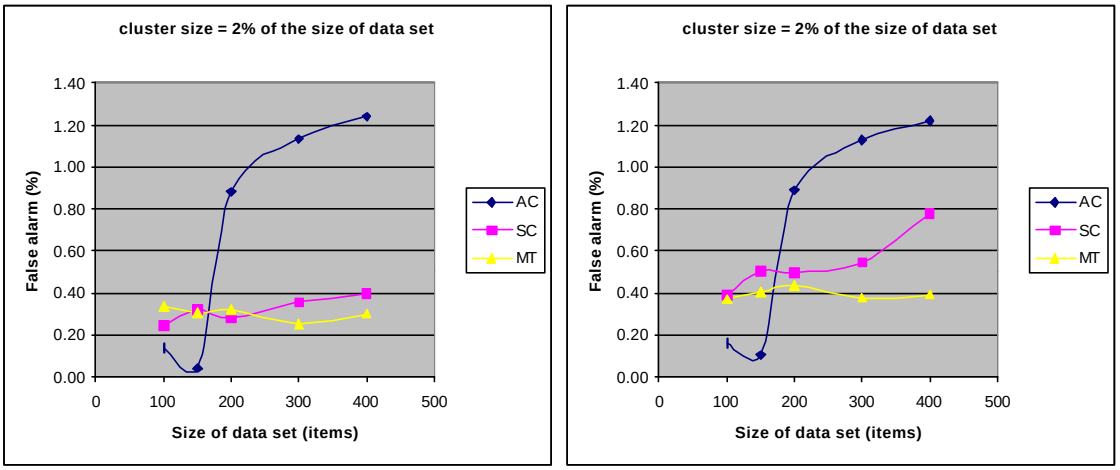

Figure 18: Number of false alarms produced by each clustering heuristic using 2% of the total data set as cluster size on "PhdThesis" (left) and "Article" (right) data sets

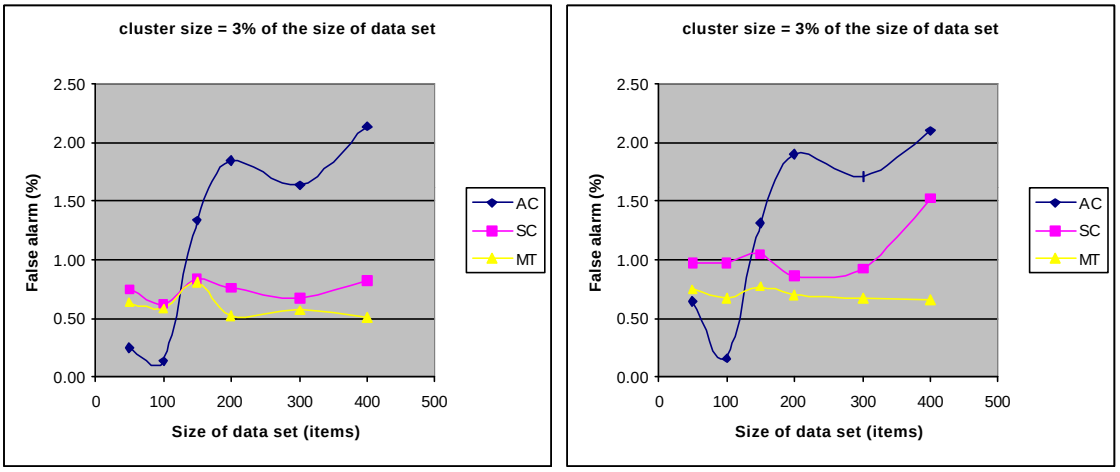

Figure 19: Number of false alarms produced by each clustering heuristic using 3% of the total data set as cluster size on "PhdThesis" (left) and "Article" (right) data sets

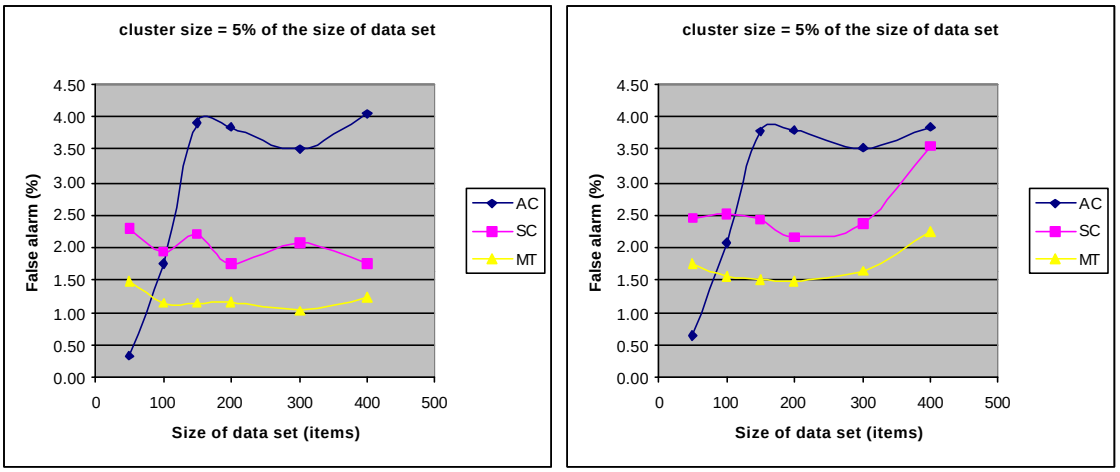

Figure 20: Number of false alarms produced by each clustering heuristic using 5% of the total data set as cluster size on "PhdThesis" (left) and "Article" (right) data sets

In the second part of analysis, we have concentrated on the accuracy of the three heuristics. Figure 18 through 20 show percentages of false alarms produced by each heuristic under different cluster sizes. The results show that, when the cluster size is less than 5% of the total data set size, the AC heuristic produces the fewest false alarms for data sets of size between 50 and 100 items. Note that when the size of data set exceeds

than 100 items, the number of false alarms produced by the AC heuristic significantly increases. Note also that on "PhdThesis" data sets the number of false alarms produced by the MT and SC seems stable when the size of the data set increases. The reason is that the data set contains elements whose duplicates are in random order and both MT and SC heuristics partition the elements based on a number of pivots that are randomly picked form the elements in the set. The AC heuristic, on the other hand, partitions the elements based on the sorted order of the distances between two elements. Such distances were obtained with respect to the order of the elements in the data set. Therefore, the probability that the elements are mismatched when using the AC heuristic is higher than that when using either MT or SC heuristic.

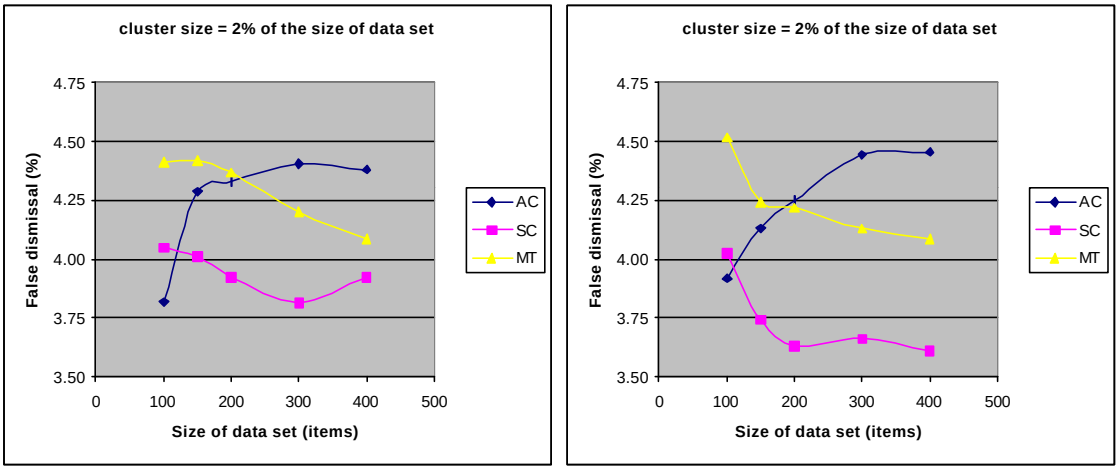

Figure 21: Number of false dismissals produced by each clustering heuristic using 2% of the total data set as cluster size on "PhdThesis" (left) and "Article" (right) data sets

Notice that, on the "Article" data set, the MT heuristic is superior in every case. On the "PhdThesis" data set, the SC heuristic sometimes produces fewer false alarms than the MT heuristic for data set sizes between 50 and 200 items when using 2% and 3% of the size of data set as cluster size. However, when the size of the data set increases,

MT tends to give a better result. In case that cluster size is 5% of the size of data set, the MT comes up ahead for every data set. This is because SC has a fixed cluster size, whereas MT allows cluster size to vary; hence, MT can produce more clusters. The more clusters, the more choices for putting an element into a cluster; thus, the number of false alarms decreases. Although increasing cluster size causes a decrease in the number of clusters, the MT heuristic can still produce more clusters when compared with the SC heuristic, which is very sensitive to the cluster size.

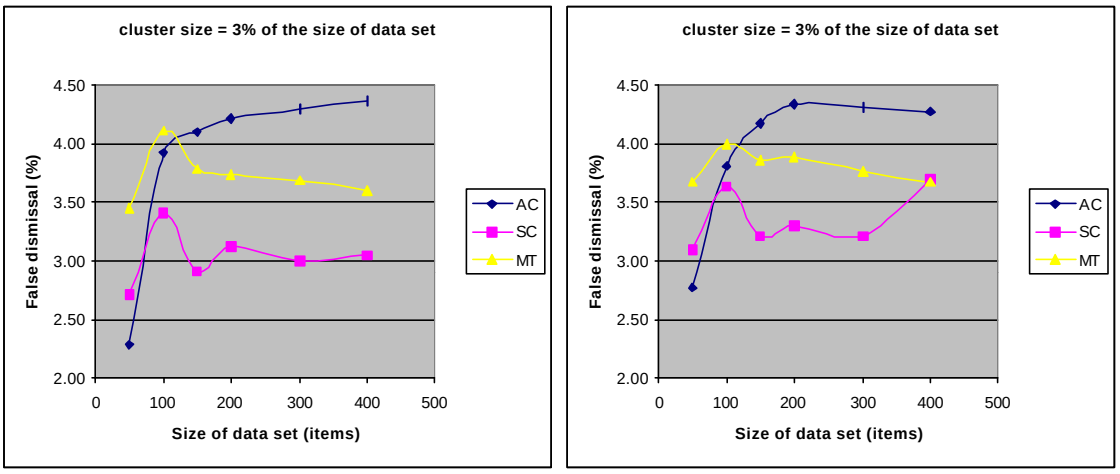

Figure 22: Number of false dismissals produced by each clustering heuristic using 3% of the total data set as cluster size on “PhdThesis” (left) and “Article” (right) data sets

Figure 21 through 23 show percentages of false dismissals produced by each heuristic under different cluster size. A false dismissal occurs when an element is placed into one cluster and its duplicate is placed into another cluster. Observe that the AC heuristic produces the fewest false dismissals for a small data set (i.e., data sets of size 100 items or less). It also produces more false dismissals than the other two heuristics on large data sets. This is because in SC and MT heuristics each element may not be compared to all duplicates of such element. For example, in SC heuristic, the element

and its duplicates were not compared to each other but to the pivots that are picked at random. In MT heuristic, each element is compared to the routing objects which may not be one of the duplicates of the element. Note that there is a higher probability of comparing the element and its duplicates when using MT heuristic than when using SC heuristic; thus, the SC heuristic produces fewer false dismissals than the MT heuristic. The SC heuristic produces the fewest false dismissals for a data set whose size is larger than 100 items.

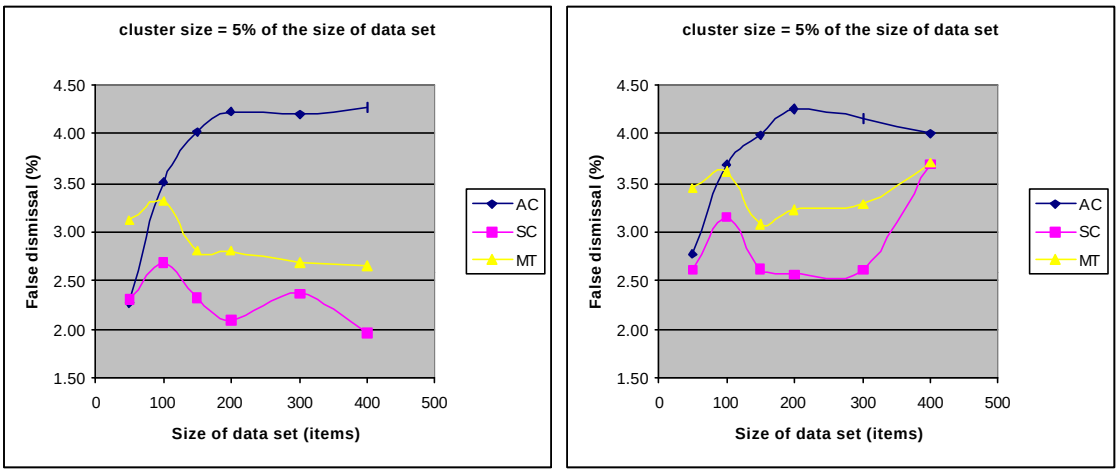

Figure 23: Number of false dismissals produced by each clustering heuristic using 5% of the total data set as cluster size on “PhdThesis” (left) and “Article” (right) data sets

Another observation is that when the size of the data set increases, the number of false dismissals increases for the AC heuristic. This is because the AC places a pair of elements into a cluster in the order of the distances between those two elements. The scenario can occur as follows: A pair of elements (e.g., the element and one of its duplicates) is placed into a non-empty cluster which fills the cluster. Other copies of the element are then forced to be in another cluster.

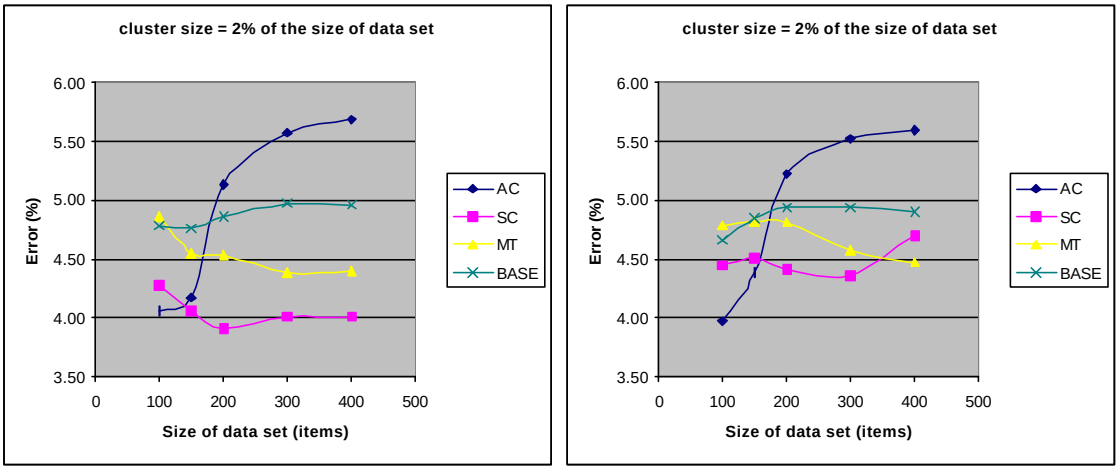

Figure 24: Matching errors for each clustering heuristic using 2% of the total data set as cluster size on "PhdThesis" (left) and "Article" (right) data sets

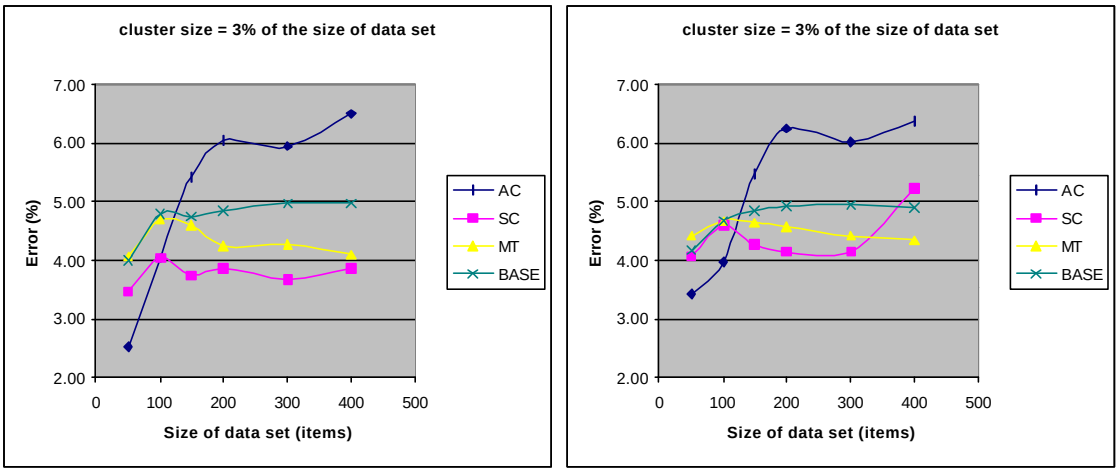

Figure 25: Matching errors for each clustering heuristic using 3% of the total data set as cluster size on "PhdThesis" (left) and "Article" (right) data sets

Figure 24 through 26 show percentages of errors produced by each heuristic on different cluster sizes. The total number of errors includes the number of false alarms and the number of false dismissals. Figure 24 through 26 have shown that, for any data set whose size is greater than 200 items, SC and MT heuristics can provide more accurate matching results than the AC heuristic. However, for a small data set (i.e., the data set of

size less than 100 items), the AC heuristic provides a more accurate result than the other two. Another observation is that both SC and MT heuristics can reduce the total numbers of errors for data sets of size larger than 100 items when the cluster size is less than the percentage of the base error. We measure base error from the number of matching errors relative to the total number of data elements with no removal. In addition, the SC heuristic performs well when the cluster size is small (e.g., 2% of the size of data set), whereas the MT heuristic performs better when the cluster size is larger (e.g., 5% of the size of data set).

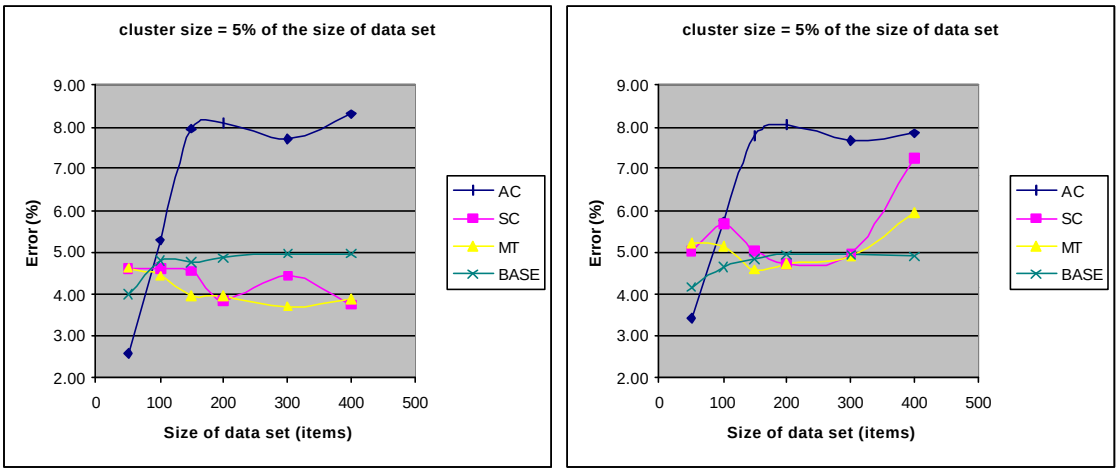

Figure 26: Matching errors for each clustering heuristic using 5% of the total data set as cluster size on “PhdThesis” (left) and “Article” (right) data sets

Figure 27 shows the number of false alarms, false dismissals and errors, which is the sum of the number of false alarms and false dismissals, when AC heuristic is used for data sets of size 50 items with variation in cluster size. According to the figure, the AC heuristic provides the most accurate matching result when the cluster size is between 3% and 5% of the size of data set. The number of errors is reduced by 18% and 36% with respect to base error for the “Article” and “PhdThesis” data sets, respectively.

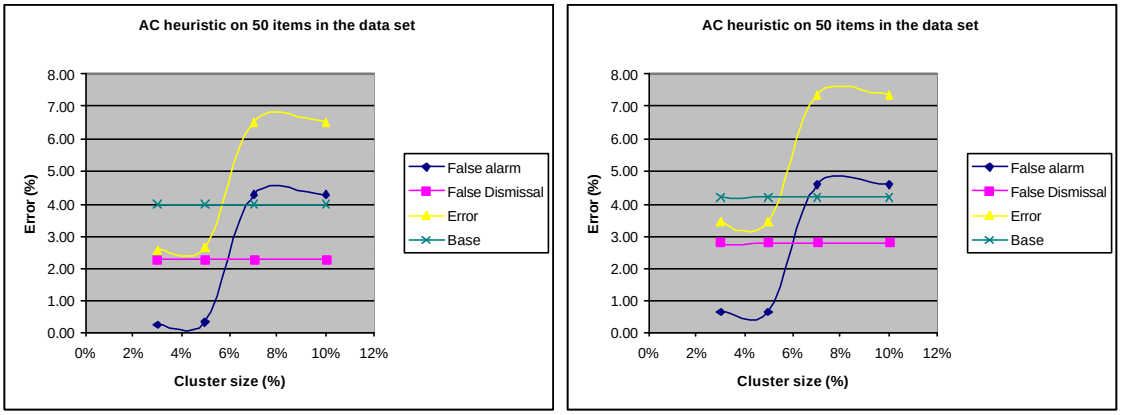

Figure 27: Matching errors produced by the All-Pair-Comparison-based heuristic with variations in the cluster size for data set of size 50 items on "PhdThesis" (left) and "Article" (right) data sets

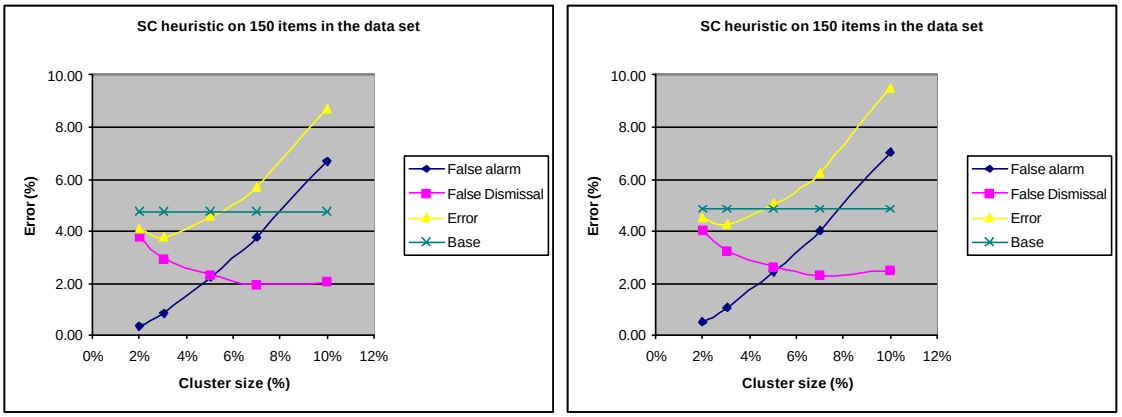

Figure 28: Matching errors produced by the Selected-Comparison-based heuristic with variations in the cluster size for data set of size 150 items on "PhdThesis" (left) and "Article" (right) data sets

Figure 28 shows the number of false alarms, false dismissals and errors, which is the sum of number of false alarms and false dismissals, when using SC heuristic for data sets of size 150 items with variation in cluster size. According to the figure, the SC heuristic provides the most accurate matching result when the cluster size is 3% of the size of data set. The number of errors is reduced by 12% and 21% with respect to the base error for the "Article" and "PhdThesis" data sets, respectively.

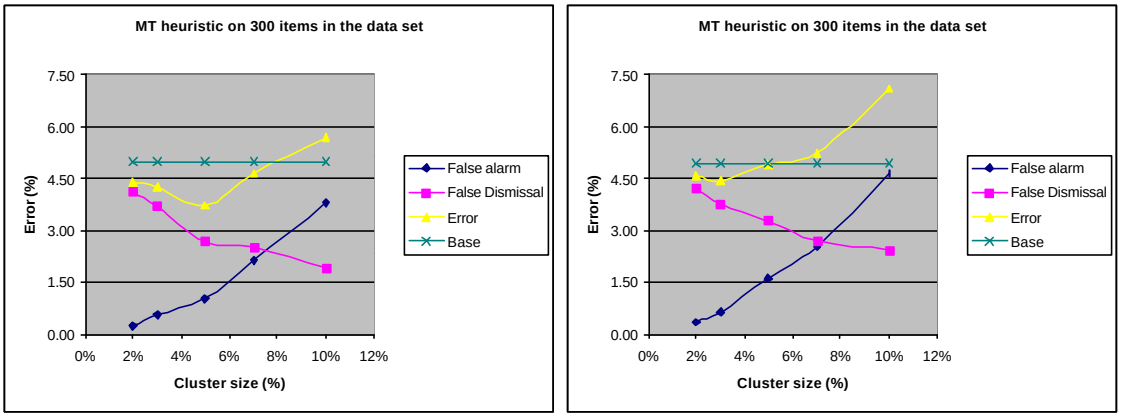

Figure 29: Matching errors produced by the M-tree-based heuristic with variations in the cluster size for data set of size 300 items on "PhdThesis" (left) and "Article" (right) data sets

Figure 29 shows the number of false alarms, false dismissals and errors, which is the sum of number of false alarms and false dismissals, when using MT heuristic for data sets of size 300 items with variation in cluster size. According to the figure, the MT heuristic provides the most accurate matching result when the cluster size is between 3% and 5% of the size of data set. The number of errors is reduced by 10% and 25% with respect to the base error for the "Article" and "PhdThesis" data sets, respectively.

In Figure 27 through 29, observe that when the cluster size increases, the number of false alarms increases as well, but the number of false dismissals decreases. The reason is that increasing the cluster size forces more elements into the same cluster, which increases the probability that an element and its duplicates are co-located. This reduces the number of false alarms. It also increases the probability that two different types of elements are co-located, which increases the number of false dismissals. The smallest number of errors is produced when the cluster size is less than 5% of the size of the data set. It is interesting to note that the base error is also about 5% of the size of data

set. Recall that the base error was measured from the number of matching errors relative to the total number of data elements with no removal. Therefore, the appropriate cluster size should be no larger than the base error. Note also that all data sets used for the experiments contain 5% duplicates. Hence, the base error can be estimated from the expected number of duplicates that can occur in the data set. For example, assume a data set that contains items that are obtained from multiple sources. If we assume that all the duplicates in the data set are from different sources (i.e., each sources provides a subset of distinct items), the expected number of duplicates is equal to the number of sources that provide the data items. Therefore, the base error is the ratio of the number of sources to the total number of data items obtained from every source. If some duplicates can be from the same source, we need statistical information about the source (e.g., the expected number of duplicates in each source) to estimate the base error. As we apply the clustering techniques to speed up the matching process at run-time, the statistics can be obtained during the built-time process. The detail of the processes at built-time and run-time are described in the next chapter.

From these results, the AC heuristic provides the best performance and accuracy when matching elements for a small data set (i.e., data sets of size no larger than 100 items). For a large data set (i.e., data sets of size larger than 300 items), MT is the preferred choice. For data sets of size between 100 and 300 items, MT performs faster than SC, but overall SC provides more accurate results. Choosing a heuristic is a trade-off between the time it takes to complete the matching and the accuracy of the result.

Based on the experimental results, our integration system supports all three heuristics for matching elements under different conditions. The system will

automatically pick the heuristic that performs best with respect to the size of the data set. However, the system also allows users to manually select a heuristic and to customize the clustering parameters such as cluster size.

CHAPTER 8
IMPLEMENTATION OF THE DATA MERGE ENGINE

The Data Merge Engine (DME) is one of the two major components located in the mediator which is a part of the middleware layer of the Integration Wizard (IWIZ) system. In chapter 4, we have already discussed the architecture of IWIZ system including the mediator and the other important system components. In this chapter, we will describe the implementation details of DME.

The Data Merge Engine is tightly-coupled with the Query Rewrite Engine (QRE), which is the focus of a related project [62]. The QRE provides the DME a query plan that specifies the sequences of join operators for combining partial results from individual sources. The task of the DME is to perform the join sequences on the partial results and to merge the restructured data from the individual sources (i.e., the restructured XML documents representing the individual, source-specific results) into one integrated result as requested by the user query. An important part of this fusion step is the reconciliation of conflicts that may exist in overlapping data that come from multiple sources.

The two main functions carried out by DME are generation of the merging specification (as a part of mediation specifications) at built-time and merging the similar data items obtained from one or more wrappers at run-time. Figure 30 and 31 show run-time and built-time control flows, respectively, between the components inside DME. Darker shaded entities represent DME components; lighter entities represent input/output for each component. In Figure 31, the Query Rewrite Engine and the wrappers are not

parts of DME. The entities shown as flowchart-documents represent XML document files. The rectangular and circular entities represent internal data structures inside DME or from other components inside the mediator (e.g., query plan and annotated ontology tree.) The block arrows represent the entry and exit points for DME.

The major run-time components of DME are the *Data Cleaner*, the *Data Joiner*, the *Cluster Generator*, the *Data Unifier*, and the *Data Filter*. The first three components are also used as built-time components. There are four more built-time components: the *Probe Query Generator*, the *Result Evaluator*, the *Parameter Adjuster*, and the *Specification Generator*. Each DME component corresponds to a Java class. The DME prototype is implemented using Java (SDK 1.3) from Sun Microsystems. Other major software tools used in our implementation are the Oracle XML Parser version 2.0.2.9 [54] and the XML-QL processor version 0.9 from AT&T Research Lab [3].

At built-time, we refer to users as special users (e.g., a system administrator) who understand well the data reconciling process. At run-time, we refer to users as font-end users who only understand interfaces for accessing data that they want.

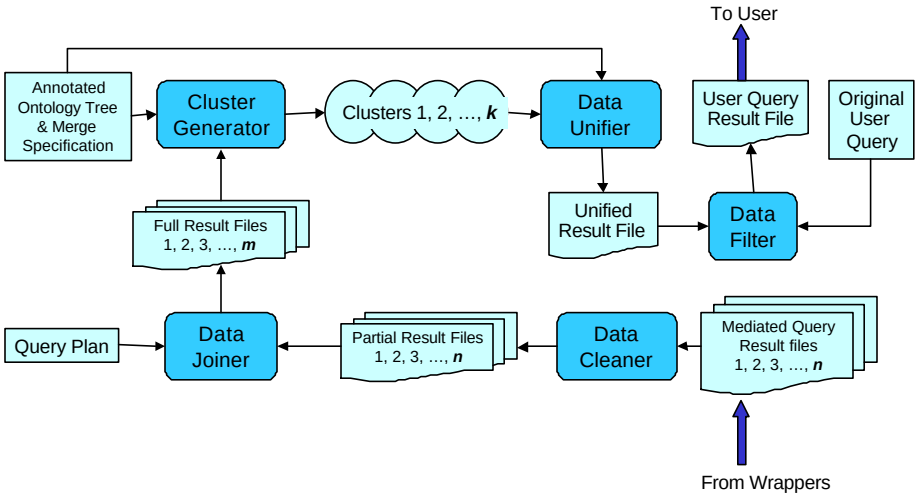

Figure 30: Run-time Control Flow among Components inside DME

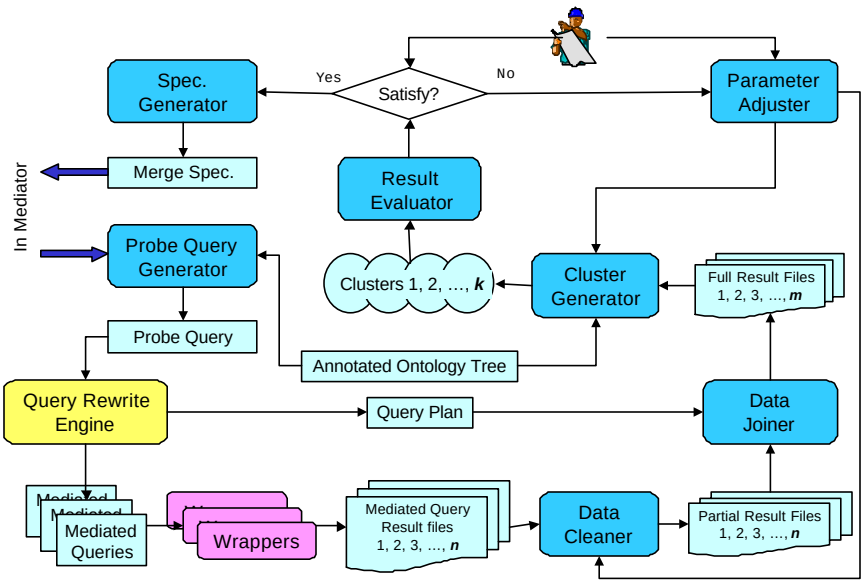

Figure 31: Built-time Control Flow among Components inside DME

Data Cleaner

When DME receives an XML document containing the result of a mediated query from a wrapper, the Data Cleaner (DC) is invoked. The DC prepares the document (e.g., removing extra white spaces) for processing by other components. Note that the wrapper only deals with the transformation of the schema, not the data contents. Since the data are from multiple sources, different sources may have different ways of specifying data values; hence, data conflicts occur. The DC provides several options for cleaning data. Users are allowed to fine tune the cleaning options as follows:

The case-sensitivity option indicates whether or not the DC should convert data value of type string into upper or lower cases. The space-removal option specifies that the DC should remove extra white spaces that may appear. The spell-check option indicates whether or not the DC should collect any data value so that their spelling

conforms to some standard dictionary. Likewise, the thesaurus option lets users specify whether or not the DC should spell-check data value using an external thesaurus. The stemming option lets users specify whether or not the DC should remove suffix or prefix in order to transform individual words into theirs root (e.g., "kindness" is transformed into "kind".) Finally, the tokenizing option lets users specify whether or not the DC should remove common function words (e.g., the, of, as, etc.) Those options are provided as a way to help users resolve synonym, acronym, and spelling conflicts.

The DC takes an XML document as an input and produces a new XML document that contains the data items whose contents have been cleaned with respect to the specified cleaning options. After that, the new XML document is forwarded to the Data Joiner.

Data Joiner

After all documents, each of which contains an answer set of data elements for a particular user query, have been cleaned, the Data Joiner (DJ) joins related data elements. To achieve this task, the DJ follows the query plan (QP) that is provided by the Query Rewrite Engine. Figure 32 shows a sample query plan. The QP contains a forest of execution trees. Each execution tree consists of a reference to a query file (e.g., "`0001.et1.xmlql`") containing a join query (e.g., an XML-QL query shown at the bottom of Figure 32) and the name of query processor (e.g., "`xmlql.cmd`") that must be invoked to execute the join query. In its current version, the DJ can invoke the XML-QL processor and processes any XML-QL queries.

The DJ takes as input a set of files each of which can contain partial results for a particular query. The DJ carries out another set of files each of which contains a set of

data elements. Each data element fully answers a particular query and may have extra information (e.g., join items) that is not needed for the result, but that was obtained during joining process. The latter set of files is forwarded to the Cluster Generator.

```
<?xml version="1.0"?>
<!DOCTYPE QueryPlan SYSTEM "QueryPlan.dtd">
<QueryPlan uquID="0001" forElement="Ontology.Bib.Article.Title">
  <ExecutionTree       queryProcessor="xmlql.cmd"
                       queryFileName="0001.et1.xmlql" />
</QueryPlan>
```

```
/* 0001.et1.xmlql */
function query()  {
WHERE
  <Ontology><Bib><Article><Title><PCDATA>$med_Title2</></></></></></>
  IN "source1.xml",
  <Ontology><Bib><Article><Title><PCDATA>$med_Title1</></></></></></>
  IN "source2.xml" ,
  $med_Title1 = $med_Title2
CONSTRUCT
  <Ontology><Bib><Article><Title><PCDATA>$med_Title1</></></></></></>
}
```

Figure 32: A sample query plan (top) and a join query (bottom) that is referenced in the query plan

Cluster Generator

The Cluster Generator (CG) collects data elements from the files containing the full (joined) result sets for the user query. It clusters those data elements and generates a clustering tree. The tree is constructed using one of the heuristics described in Chapter 6. Based on our experimental results, an appropriate heuristic is chosen with respect to the number of data elements to be clustered.

The CG creates a clustering tree. Each leaf of the tree represents a cluster containing a set of data elements that are tentatively equivalent (i.e., represents the same real world element). During the built-time phase, every cluster is provided to a human via the Result Evaluator to estimate errors of matching result. During the run-time

phase, every cluster is provided to the Data Unifier who is responsible for fusing data items that are in the same cluster.

Data Unifier

Based on the assumption that data elements in the same cluster are tentatively equivalent, the Data Unifier (DU) fuses those elements together using the threshold and the constraint information, both obtained from the annotated ontology tree. Let E_i be the i^{th} element of type E. All elements in the same cluster are being unified. Let S_i be an element of type S and be a subelement of E_i. The data unifying process is as follows:

1. For all subelements S_i of E_i whose constraint is *exactly-one*, if S_i contains a value, count the frequency for all possible value for S_i. Then, pick the element that has the highest frequency. If there is a tie and S_i is a complex element, pick one of the subelements, say S_j, of E_j such that E_j contains more information than other elements E_i where $i \neq j$. Element E_j is selected as a base element. If there is a tie and S_i is not a complex element, randomly pick one of the tie elements. This is based on the assumption that the elements containing more information tend to be more complete.

2. For all subelements S_i of E_i whose constraint is *zero-or-one*, if there is no such subelement, do nothing. If there is only subelements, place it into the result element. Otherwise, select subelement S_j from the base element E_j. If there is no such element, select subelement S_k from element E_k such that E_k is the closest element to E_j.

3. For all subelements S_i of E_i whose constraint is *one-or-more* or *zero-or-more*, compare all of them to subelements S_j of the base element E_j. For any pair (S_i, S_j), if the distance between them is greater than or equal to the given threshold, which can be obtained from the annotated ontology tree, place S_i into the result element. This is based on the assumption that S_i is different from S_j. If the distance between them is less than the threshold, discard S_j.

4. Repeat steps 1 – 3 recursively on any complex element.

Figure 33 illustrates the result of unifying two `Book` elements. The two `Book` elements are at the top left and right corner of the figure, and the resulting `Book` element is at the bottom. Let us assume that each `Book` can have only one `Title`. Note that the `Title`s of the two `Book`s are not identical. One of the two `Title`s must be chosen. We pick the `Title` of the top right `Book` since this `Book` contains more information than the other. Therefore, the top right `Book` is chosen as a base element.

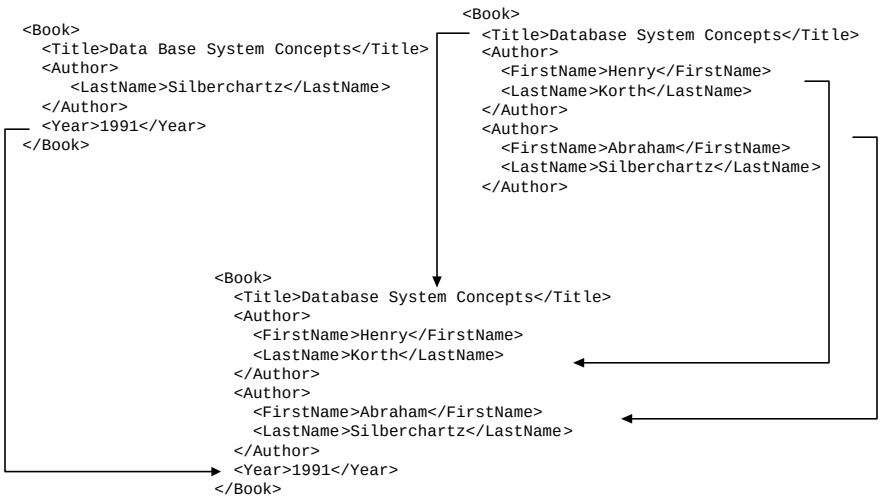

Figure 33: Conceptual unification of two incomplete book instances into one

Let us assume that each Book can have *zero-or-one* Year subelement. Since the top right Book does not have a Year whereas the top left Book does, the Year of the second Book is chosen. If both two Books have Year, the Year of the top right Book will be chosen since the top right Book was chosen as a base. Let us assume that each Book can have *one-or-more* Authors. We compare all Authors of the first Book against all Authors of the base element Book. Pick the Author of the base element Book and others Authors whose distance is greater than the threshold. In this scenario, there exists no other Authors whose distance is greater than the threshold.

The Data Unifier takes as input a set of clusters, each of which contains data elements, and produces a set of the unified elements. Those unified elements are provided to the Data Filter.

Data Filter

Each unified element obtained from the Data Unifier may contain extra information which is non-requested information that is used for joining elements. This extra information is obtained from the sources to be able to form the full result for the answer set; hence, this extra information needs to be filtered out to provide the correct result. To achieve this goal, the original input query to the mediator is needed. The Data Filter only invokes a query processor to execute the original input query against the set of unified elements. Currently, since the user query is an XML-QL query, the DME invokes the XML-QL processor to execute the query against the set of unified elements that are stored in an XML document.

Result Evaluator

The Result Evaluator (RE) is invoked during the built-time phrase after the Cluster Generator has produced a set of clusters. Using a GUI, the RE provides the result of clustering data elements and the values of parameters for data cleaning and clustering. The interface allows users to evaluate and verify the clustering process by moving the data elements from one cluster to another. Then, the RE automatically calculates the percentage error of the result compared to the changes made by users. Users can choose not to modify the parameters of the hierarchical clustering model if the percentage of errors is acceptable; and the values of parameters will be stored in a persistent storage by the Specification Generator and used in the clustering process at run-time. We refer to the values of parameters as a *Merge Specification*. However, users can choose to browse and change the value of any of those parameters. After browsing and changing those parameters, the Parameter Adjuster is invoked to internally modify the value of parameters accordingly and to restart the clustering process using the new values of parameters.

Parameter Adjuster

After the result of element matching has been evaluated by users via the Result Evaluator interface, and the users decide to change the value of parameters of the clustering process, the Parameter Adjuster (PA) is invoked. The PA maintains the history of the parameter adjustments. The history is used for references and allows users to backtrack and fine-tune the parameters. After the parameters have been modified, the PA will invoke either the Cluster Generator or the Data Cleaner depending on the components that the modified parameters involve. If a parameter for Data Cleaner is

changed, the process will restart at the cleaning step. If a parameter for Cluster Generator is changed, the process will restart at the clustering step.

Specification Generator

When the clustering is completed and verified by the users, the Specification Generator is invoked to store the values of clustering parameters based on the input from the users. We refer to those parameter values as *Merge Specification* (MS). The MS is a part of the mediation specification (as shown in Figure 7) that contains information about sources (e.g., location, availability, and reliability) used by the Query Rewriting Engine; it also contains other information used by the Data Merge Engine for reconciling data items from multiple sources. We implement a data structure to store the MS and a list of concepts for users to browse and to create a query based on those concepts. Such structure is called the *annotated ontology tree*. The MS contains information that will be used for reconciling data items. The information is associated with nodes and edges inside the annotated ontology tree. Such information consists of a set of parameters for data cleaning, data clustering, and data unifying. The data cleaning parameters contain information regarding case-sensitivity, space-removal, spell-check, thesaurus, stemming, and tokenizing. The data clustering parameters are weight values that are used during the clustering process as a factor to determine the distance between two elements. The data unifying parameters are threshold values that are used during the unifying process.

Probe Query Generator

A very important built-time process inside the DME is the generation of probe queries. A probe query is generated with respect to a concept defined in the ontology. The query simulates an input query that will be handled by the Query Rewrite Engine.

The purpose of generating a probe query is to poll from each wrapper a set of sample data as well as the information about the sources such as the availability and reliability of each source (i.e., which underlying source contains which subsets of the hierarchical concepts defined in the ontology, and which source is the most reliable if the sources contain overlapping data). We refer to the sample sets returned from wrappers as training sets. A training set is a set of the restructured XML documents. After receiving the training set from all wrappers, the built-time merging process continues. The Data Cleaner and other built-time components are invoked. The built-time merging process produces a draft of the merge specification which is validated by the users. The process finishes with the final version of the merge specification that has been verified. The process may contain multiple transactions, each of which is for each probe query, and may enclose multiple iterations due to the user inputs during the verification of the merge specification. However, the built-time merge process is carried out once for a given combination of source and ontology schemas.

The number of probe queries that are generated by the Probe Query Generator (PG) is bounded by the number of concepts defined in the global ontology. As we mentioned, a probe query simulates an input query. However, the difference between the probe query and the actual input query is that the actual input query is not required to contain information for sampling data sources. Such information could speed up the built-time process, since it could reduce the size of the result from each wrapper. The sampling parameters involve the size of the training set. The appropriate values of the parameters should be indicated so that the merge specification can be generated properly. Although the PG creates a probe query with proper sampling parameters, the sampling

process should be performed at the wrappers since the wrappers are tightly-coupled with the sources.

Currently, the PG only generates a probe query without sampling parameters. We plan to provide a full function of supporting a sampling query in the next version of our integration system (i.e., IWIZ system). To achieve the goal, two major tasks must be carried out. First, the query language parser must be modified so that it can correctly interpret the query with sampling parameters. Second, the data-sampling component should be added inside the wrapper.

CHAPTER 9
CONCLUSTION

In the integration and management of modern information systems, overcoming data heterogeneities efficiently and automatically is a major, ongoing concern. Heterogeneities can arise due to schematic and semantic differences that exist among related data items in different sources. The intent of this dissertation is to advance the state-of-the-art in automatic conflict resolution and reconciliation.

In this dissertation, we have addressed the problem of merging similar and overlapping data items from multiple information sources. We have presented a hierarchical clustering model for object reconciliation that semi-automatically merges similar and overlapping information. The methodology addresses the general object reconciliation problem – a real-world object can be represented in many different ways in different sources [40]. Our model is based on a set of clustering trees and a set of distance functions. A distance function specifies the semantic distance between any pair of objects. A clustering tree contains a set of clusters each of which contains a set of objects that are related to each other. We have described a framework for designing optimal clustering trees. The framework takes into account the characteristics of the data items to be reconciled (e.g., the data distribution), the domain spaces (e.g., Euclidean and Metric), and the characteristics of the tree (e.g., balanced/unbalanced, size of the cluster, overlapping of clusters, split/decomposition strategies). We have implemented several heuristics – *All-Pair Comparison, Selected Comparison* and *M-tree* – to create clustering

trees. Our experimental results have verified our expectations that different heuristics are suited for different characteristics of data sets (e.g., the size of the data set and the expected number of duplicate objects). Given the maximum cluster size with respect to the expected number of duplicate objects in the data set, all three heuristics perform very well in terms of computation time and provide accurate results with respect to object matching. Among the three heuristics, for a small data set (e.g., data set containing no more than 100 data items), the All-Pair-Comparison-based heuristic generates the most accurate results. For a large data set (e.g., the data set containing more than 300 data items), the M-tree-based heuristic is the fastest in obtaining the most accurate results. For a medium data set (e.g., the data set containing between 100 and 300 data items), the Selected-Comparison-based heuristic provides the most accurate results.

Our research is couched within the Integration Wizard (IWIZ) system. The system allows end-users to access and retrieve information from multiple sources through a consistent, integrated view. To enhance performance, the system uses a combined mediation/data warehousing approach to information integration.

Our hierarchical clustering model is implemented inside the Data Merge Engine (DME), an integral component of the IWIZ mediator, which constitutes the middle ware. Given the popularity of the Web, we focus on reconciling XML-based, semistructured data. The DME takes as input a set of XML documents from underlying wrappers. Each document contains a set of data items from each source. The DME uses the clustering model to discover related data items from multiple sources. Related data items are unified and returned as the query result.

We assume that the data from the sources may be incomplete and contain errors, e.g., inconsistencies, misspellings, etc. We have classified these potential conflicts into three categories [56]: (1) Structural conflicts, which arise when the schemas of the data sources exhibit discrepancies, (2) domain conflicts, which arise when the schemas and domains of the data source exhibit discrepancies, and (3) data conflicts, which arise when the semantic of the data sources exhibit discrepancies

Contributions

This dissertation work contributes to the state-of-the-art in information integration in the following four ways. First, we specify a new classification for structural and semantic conflicts among XML-based data sources [56]. Second, we present a hierarchical clustering model as a new solution to speed up the merging of similar and overlapping information. We design and implement a number of heuristics for constructing the clustering tree inside our clustering model and provide a distance function for measuring the semantic distance between objects in a cluster. Third, we perform qualitative analysis of the performance of the hierarchical clustering model including the distance functions and provide the optimal values of the parameters for constructing the hierarchical structure. The evaluation is based on (a) time complexity of the algorithms using the heuristics, (b) the number of false alarms that arise when two elements are matched although they are unmatched in the real-world situation, and (c) the number of false dismissals that arise when two elements representing the same real-world element are mismatched. Finally, we design and implement a prototype of the IWIZ information integration system. The design and implementation of the Data Merge Engine is described in Chapter 8. Other IWIZ components have been developed and

being developed by other colleagues in Database Research and Development Center at University of Florida. All IWIZ components communicate to one another using a set of Remote Method Invocation (RMI) interfaces

Future Challenges

The concept of clustering data items is relatively new in the context of data integration. To provide a theoretical underlying support, a formal description of the hierarchical clustering model for data reconciliation can be expanded. In addition, although our integration system using the clustering model currently supports several heuristics for constructing the clustering tree, we can extend the system and make it more scalable and flexible for users by providing more varieties of heuristics. By using heuristics that the system currently provides, the clustering tree is built under the assumption that the tree structure and all data items to be reconciled can be fit into memory. Therefore, the future works should take into account reconciling data items on a disk or other secondary storages.

In the current situation, the Data Merge Engine (DME) contains one-step (supervised) learning capability in such a way that, at built-time, it allows a user to verify the merge specification that contains values of parameters for cleaning, dustering and unifying data items, after the DME merges data items with an initial value of each parameter. We can extend the learning capability to DME by keeping track of user feedback for each query result and using such information to automatically adjust the values of parameters for merging data items.

During built-time phase of DME, the Probe Query Generator creates a set of probe queries based on concepts in ontology. The probe queries are sent to wrappers to

poll a set of sample data as well as to obtain information about sources such as availability and reliability of each source. In the current situation, each wrapper returns the mediator a full set of data as a query result that answers a probe query; the returning of full data set involves the communication time between wrappers and mediator as well as the memory (or disk) space that is used in the mediator to merge data items from multiple sources. To reduce the communication time and memory space, we can extend our integration system to support data sampling. In other words, a subset of data from each wrapper to mediator, called a *training* set, is returned. Data sampling should occur in both mediator and wrappers. In the mediator, a probe query is created and attached with sampling parameters that specify percentage of returned data items relative to the total size of data set. In each wrapper, the sampling parameters is interpreted; the probe query is executed; a training set is obtained based on the value of sampling parameters and the training set is returned to the mediator.

In the current situation, our integration system prototype is focusing on XML-based semistructured data. The prototype can be extended to support more varieties of data such as relational or object oriented data. All the data items are currently represented in an XML document and their schema is defined in a DTD. Due to the rapid evolving of XML and its related technology, the next version of the prototype should be aware of the trend of the XML technologies. Most of the XML technologies are carried out in form of recommendations from W3C, the leading organization coordinating the research and development efforts of the XML community.

Last but not least, XML query processing is a very important issue. The current query processor in IWIZ is built on the top of XML-QL processor. The difficulty of

learning how to query XML data using XML-QL was realized during the development of our system prototype. At the time of writing, many proposals involving different XML query languages have been submitted and are being reviewed by W3C. Therefore, the next version of IWIZ may need to be able to support a different query language depending on the recommendation of the W3C.

APPENDIX A
DOCUMENT TYPE DEFINITION OF THE BIBLOGRAPHY ONTOLOGY

Ontology.dtd

```
<!--  IWIZ's ONTOLOGY in one single Integrated DTD -->

<!ELEMENT Ontology (Bib)*>

<!ELEMENT Bib  (Article | Book | Booklet | InBook  | InCollection |
                InProceedings | Manual | MastersThesis | PhdThesis |
                Misc | Proceedings | TechReport | UnPublished )*>

<!ELEMENT Article     ( Author+,  Title, Year, Month?,  Pages?, Note?,  Journal  )>

<!ELEMENT Book       ((Author+ | Editor+),   Title,     Publisher,
                        Year,      Month?,   (Volume  |  Number1)?,
                        Series?,   Address?,  Edition?,  ISBN,  Cost?,     Note? )>

<!ELEMENT Booklet   ( Author*,  Title,  HowPublished?, Year?, Month?,
                        Address?, Note?  )>

<!ELEMENT InBook    ( Author+,   Title,    Year,      Month?,
                        Publisher, (Pages | Chapter?), Book  )>

<!ELEMENT InCollection     ( Author+,   Title,     Year,      Month?,
                           (Pages | Chapter?),    Type?,     Note?,  Collection )>

<!ELEMENT InProceedings ( Author+, Title,  Year,  Month?,
                             (Pages | Chapter?),    Note?,     Proceedings )>

<!ELEMENT Manual       ( Title, Author*,   Year?, Month?,
                        Edition?,  Address?,  Organization?, Note?   )>

<!ELEMENT MastersThesis  ( Author,  Title,  School, Address?,
                             Year, Month?, Type?, Note? )>

<!ELEMENT Misc       ( Author?, Title?, HowPublished?, Year?, Month?, Note?)>

<!ELEMENT PhdThesis        ( Author, Title, School,  Address?,
                             Year, Month?, Type?, Note?   )>
```

```
<!ELEMENT TechReport     ( Author+,   Title,     Institution, Year, Month?,
                           Type?, Number1?,  Address?,  Note? )>

<!ELEMENT UnPublished    ( Author+, Title,  Year?,  Month?,  Note )>

<!ELEMENT Address (#PCDATA)>

<!ELEMENT Author (Firstname?, Lastname, Address?)>

<!ELEMENT Chapter (#PCDATA) >

<!ELEMENT Collection ((Author+ | Editor+), Title, Publisher?)>

<!ELEMENT Cost (#PCDATA)>

<!ELEMENT Editor (#PCDATA)>

<!ELEMENT Edition (#PCDATA)>

<!ELEMENT Firstname (#PCDATA)>

<!ELEMENT ISBN (#PCDATA)>

<!ELEMENT HowPublished (#PCDATA) >

<!ELEMENT Institution (#PCDATA) >

<!ELEMENT Journal (Title, Year?, Month?, Volume?, Number1? )>

<!ELEMENT Lastname (#PCDATA)>

<!ELEMENT Month (#PCDATA) >

<!ELEMENT Note (#PCDATA) >

<!ELEMENT Number1 (#PCDATA) >

<!ELEMENT Organization (#PCDATA) >

<!ELEMENT Pages (#PCDATA) >

<!ELEMENT Proceedings    ( Title, Editor*,  Year?, Month?, (Volume | Number1)?,
                           Series?, Address?, Organization?, Publisher?, Note?   )>

<!ELEMENT Publisher (#PCDATA) >
```

<!ELEMENT School (#PCDATA) >

<!ELEMENT Series (#PCDATA)>

<!ELEMENT Title (#PCDATA) >

<!ELEMENT Type (#PCDATA) >

<!ELEMENT Volume (#PCDATA) >

<!ELEMENT Year (#PCDATA) >

APPENDIX B
EXPERIMENTAL RESULTS

Table B-1: Base error on each data set

Data set	Size of data set (items)	Number of variation in spelling of data values in the data set	Base error (in %)
Article	50	1	4.33
Article	100	1	5.54
Article	150	1	4.98
Article	200	1	4.85
Article	400	1	5.00
Article	50	2	4.16
Article	100	2	4.67
Article	150	2	4.85
Article	200	2	4.93
Article	300	2	4.94
Article	400	2	4.90
Article	50	4	3.67
Article	100	4	5.01
Article	150	4	4.75
Article	200	4	4.97
Article	400	4	4.93
Article	50	7	4.00
Article	100	7	4.77
Article	150	7	4.93
Article	200	7	4.92
Article	400	7	5.04
PhdThesis	50	2	4.00
PhdThesis	100	2	4.79
PhdThesis	150	2	4.76
PhdThesis	200	2	4.86
PhdThesis	300	2	4.98
PhdThesis	400	2	4.96

Table B-2: Results on "Article" data sets for all-pair-comparison-based (AC), selected-comparison-based (SC), and M-tree-base (MT) heuristics with uniform weights using terms that are separated by no more than *one* variation in the spelling

Test Description	Time (sec)			False alarm (%)			False dismissal (%)			Error (%)		
	AC	SC	MT	AC	SC	MT	AC	SC	MT	AC	SC	MT
50 items 3% cluster size	7	6	3	0.57	1.10	0.69	2.86	3.39	3.71	3.43	4.49	4.41
50 items 5% cluster size	7	4	3	0.57	2.43	1.65	2.86	2.76	3.33	3.43	5.18	4.98
50 items 7% cluster size	7	3	3	4.16	4.37	2.63	2.53	2.73	3.00	6.69	7.10	5.63
50 items 10% cluster size	7	3	3	4.16	6.69	3.35	2.53	2.86	2.55	6.69	9.55	5.90
50 items 15% cluster size	7	2	2	12.00	12.18	6.20	2.53	2.71	1.82	14.53	14.90	8.02
50 items 20% cluster size	6	1	2	16.49	16.51	7.63	2.45	2.47	1.12	18.94	18.98	8.76
100 items 2% cluster size	27	26	10	0.08	0.35	0.35	4.63	4.87	5.21	4.71	5.22	5.56
100 items 3% cluster size	26	17	7	0.10	0.76	0.77	4.63	4.30	4.99	4.73	5.06	5.76
100 items 5% cluster size	27	10	6	1.78	2.00	1.84	4.28	3.49	4.74	6.06	5.49	6.58
100 items 7% cluster size	27	7	5	5.41	4.12	3.18	4.04	2.74	3.91	9.45	6.86	7.09
100 items 10% cluster size	27	5	5	7.52	6.79	4.26	3.96	3.23	3.16	11.47	10.02	7.41
100 items 15% cluster size	27	4	5	11.07	10.64	7.06	3.72	2.54	2.70	14.79	13.18	9.75
100 items 20% cluster size	27	2	4	16.69	16.39	11.77	3.03	2.73	3.54	19.72	19.12	15.31
150 items 2% cluster size	65	41	13	0.05	0.45	0.37	4.36	4.08	4.45	4.41	4.52	4.81
150 items 3% cluster size	65	24	12	1.09	1.07	0.68	4.07	3.36	3.81	5.16	4.43	4.48
150 items 5% cluster size	65	16	10	3.49	2.38	1.22	3.82	2.72	2.87	7.31	5.10	4.09
150 items 7% cluster size	65	12	9	4.66	4.01	1.89	3.60	2.40	2.17	8.26	6.42	4.07
150 items 10% cluster size	64	8	10	7.10	6.89	3.62	3.53	2.47	2.03	10.62	9.36	5.65
150 items 15% cluster size	64	6	7	12.54	11.55	7.20	3.74	2.35	1.87	16.28	13.89	9.07
150 items 20% cluster size	63	4	7	17.69	16.71	9.96	3.20	2.22	2.02	20.89	18.93	11.98
200 items 2% cluster size	110	54	17	0.74	0.46	0.36	4.11	3.81	4.22	4.85	4.27	4.58
200 items 3% cluster size	110	36	16	1.68	0.96	0.71	4.06	3.32	3.94	5.74	4.28	4.64
200 items 5% cluster size	110	21	14	3.45	2.57	1.78	3.83	2.90	3.35	7.28	5.46	5.12
200 items 7% cluster size	110	15	13	5.43	4.05	2.99	3.80	2.00	3.02	9.23	6.05	6.01
200 items 10% cluster size	109	11	13	8.17	6.99	3.71	3.57	2.30	1.99	11.73	9.30	5.70
200 items 15% cluster size	109	7	10	12.53	11.43	7.53	3.31	2.21	1.95	15.84	13.64	9.48
200 items 20% cluster size	110	5	7	17.71	16.94	13.90	3.17	2.20	3.30	20.88	19.14	17.19
400 items 2% cluster size	416	102	33	1.05	0.62	0.45	4.31	3.86	4.23	5.35	4.48	4.68
400 items 3% cluster size	416	70	33	1.85	1.23	0.76	4.12	3.49	3.75	5.98	4.71	4.51
400 items 5% cluster size	414	42	31	3.67	2.51	1.91	3.91	2.75	3.49	7.58	5.27	5.41
400 items 7% cluster size	414	28	31	5.60	5.01	2.61	3.93	3.08	2.66	9.54	8.10	5.28
400 items 10% cluster size	415	21	24	8.42	7.56	4.83	3.64	2.79	2.97	12.06	10.35	7.80
400 items 15% cluster size	415	14	18	12.72	11.89	8.36	3.43	2.60	2.81	16.15	14.49	11.17
400 items 20% cluster size	418	9	16	18.02	16.76	11.29	3.22	1.96	2.73	21.24	18.72	14.02

Table B-3: Results on "Article" data sets for all-pair-comparison-based (AC), selected-comparison-based (SC), and M-tree-base (MT) heuristics with uniform weights using terms that are separated by no more than *two* variations in the spelling

Test Description	Time (sec)			False alarm (%)			False dismissal (%)			Error (%)		
	AC	SC	MT	AC	SC	MT	AC	SC	MT	AC	SC	MT
50 items 3% cluster size	7	7	5	0.65	0.98	0.76	2.78	3.10	3.67	3.43	4.08	4.43
50 items 5% cluster size	7	5	4	0.65	2.45	1.76	2.78	2.61	3.45	3.43	5.06	5.20
50 items 7% cluster size	7	3	3	4.57	4.29	2.69	2.78	2.49	3.08	7.35	6.78	5.78
50 items 10% cluster size	7	3	3	4.57	6.27	3.78	2.78	2.27	2.80	7.35	8.53	6.57
50 items 15% cluster size	7	2	2	12.41	11.73	9.14	2.78	2.10	1.92	15.18	13.84	11.06
50 items 20% cluster size	7	1	2	16.49	16.51	9.47	2.29	2.31	2.00	18.78	18.82	11.47
100 items 2% cluster size	27	27	10	0.16	0.39	0.37	3.82	4.05	4.41	3.98	4.44	4.78
100 items 3% cluster size	27	18	8	0.16	0.97	0.68	3.82	3.64	4.01	3.98	4.61	4.68
100 items 5% cluster size	27	10	7	2.06	2.53	1.55	3.70	3.16	3.62	5.76	5.69	5.16
100 items 7% cluster size	27	7	5	5.82	5.25	3.38	3.58	3.01	2.93	9.39	8.25	6.32
100 items 10% cluster size	27	5	6	7.84	7.02	4.03	3.41	2.60	2.51	11.25	9.62	6.53
100 items 15% cluster size	27	4	5	11.66	11.74	7.55	3.43	2.77	2.09	15.09	14.52	9.64
100 items 20% cluster size	27	2	4	17.66	17.41	11.13	3.13	2.89	2.06	20.79	20.30	13.19
150 items 2% cluster size	63	42	15	0.11	0.50	0.41	4.29	4.01	4.42	4.39	4.51	4.82
150 items 3% cluster size	64	24	13	1.32	1.05	0.78	4.17	3.22	3.86	5.49	4.27	4.64
150 items 5% cluster size	64	16	13	3.79	2.42	1.51	3.99	2.63	3.07	7.78	5.05	4.58
150 items 7% cluster size	64	11	10	5.25	4.00	2.65	4.06	2.26	2.85	9.32	6.26	5.50
150 items 10% cluster size	63	8	10	7.49	7.03	4.48	3.79	2.49	2.75	11.28	9.52	7.23
150 items 15% cluster size	63	6	7	12.55	11.68	7.38	3.62	2.36	2.05	16.17	14.04	9.43
150 items 20% cluster size	63	4	7	18.19	16.85	10.81	3.58	2.23	2.20	21.77	19.08	13.01
200 items 2% cluster size	109	53	17	0.89	0.49	0.44	4.33	3.92	4.37	5.22	4.42	4.81
200 items 3% cluster size	109	35	16	1.90	0.86	0.70	4.34	3.30	3.89	6.24	4.16	4.59
200 items 5% cluster size	109	21	15	3.80	2.16	1.49	4.26	2.57	3.23	8.07	4.72	4.72
200 items 7% cluster size	109	15	14	5.60	4.76	2.59	4.11	2.78	2.74	9.71	7.54	5.33
200 items 10% cluster size	109	10	14	8.42	6.79	3.84	3.90	2.17	2.24	12.32	8.96	6.08
200 items 15% cluster size	108	7	9	12.89	11.36	8.40	3.75	2.22	2.97	16.64	13.58	11.38
200 items 20% cluster size	109	5	8	17.83	16.75	10.89	3.36	2.08	2.14	21.19	18.83	13.04
300 items 2% cluster size	256	84	28	1.13	0.55	0.38	4.40	3.81	4.20	5.53	4.36	4.58
300 items 3% cluster size	257	57	26	1.71	0.93	0.67	4.31	3.21	3.76	6.02	4.14	4.43
300 items 5% cluster size	257	31	25	3.53	2.36	1.64	4.15	2.61	3.29	7.68	4.97	4.92
300 items 7% cluster size	257	22	23	5.87	4.37	2.54	3.91	2.41	2.69	9.77	6.79	5.23
300 items 10% cluster size	258	16	20	8.50	7.33	4.66	3.80	2.57	2.45	12.30	9.90	7.10
300 items 15% cluster size	259	11	14	12.58	12.03	7.81	3.46	2.75	2.20	16.04	14.78	10.02
300 items 20% cluster size	259	7	12	17.86	17.00	12.47	3.20	2.20	2.79	21.06	19.20	15.26
400 items 2% cluster size	437	108	36	1.22	0.78	0.39	4.38	3.93	4.09	5.60	4.70	4.48
400 items 3% cluster size	439	70	34	2.10	1.53	0.67	4.28	3.70	3.68	6.38	5.22	4.34
400 items 5% cluster size	437	42	31	3.84	3.55	2.25	4.01	3.69	3.70	7.86	7.24	5.95
400 items 7% cluster size	435	28	33	5.77	5.55	2.73	3.98	3.52	2.84	9.76	9.07	5.58
400 items 10% cluster size	436	21	24	8.39	8.35	5.36	3.57	3.48	3.31	11.97	11.83	8.67
400 items 15% cluster size	435	14	18	12.86	12.74	9.13	3.48	3.36	3.21	16.33	16.10	12.34
400 items 20% cluster size	439	9	16	18.03	17.88	13.17	3.23	2.99	3.05	21.26	20.87	16.22

Table B-4: Results on "Article" data sets for all-pair-comparison-based (AC), selected-comparison-based (SC), and M-tree-base (MT) heuristics with uniform weights using terms that are separated by no more than *four* variations in the spelling

Test Description	Time (sec)			False alarm (%)			False dismissal (%)			Error (%)		
	AC	SC	MT	AC	SC	MT	AC	SC	MT	AC	SC	MT
50 items 3% cluster size	7	7	4	0.73	1.16	0.90	2.45	2.80	3.33	3.18	3.96	4.22
50 items 5% cluster size	7	5	3	0.82	2.31	1.88	2.45	1.98	2.96	3.27	4.29	4.84
50 items 7% cluster size	7	4	3	4.73	4.63	2.90	2.45	2.35	2.73	7.18	6.98	5.63
50 items 10% cluster size	7	3	3	4.73	6.67	4.12	2.45	2.18	2.47	7.18	8.86	6.59
50 items 15% cluster size	7	2	2	12.57	12.16	7.24	2.45	2.04	1.76	15.02	14.20	9.00
50 items 20% cluster size	7	1	2	16.41	16.65	10.33	2.45	1.96	1.53	18.86	18.61	11.86
100 items 2% cluster size	31	30	11	0.32	0.39	0.35	4.34	4.39	4.74	4.67	4.78	5.10
100 items 3% cluster size	31	20	8	0.36	0.77	0.72	4.34	3.78	4.45	4.71	4.55	5.18
100 items 5% cluster size	31	12	7	2.12	2.41	1.58	4.10	3.38	3.85	6.22	5.80	5.43
100 items 7% cluster size	31	8	6	5.96	4.67	2.85	4.06	2.77	2.82	10.02	7.43	5.67
100 items 10% cluster size	31	6	6	7.94	6.72	4.02	4.04	2.64	2.89	11.98	9.36	6.90
100 items 15% cluster size	31	4	6	12.14	11.56	6.80	4.00	2.93	2.31	16.14	14.48	9.11
100 items 20% cluster size	31	3	5	17.58	16.83	9.71	3.78	2.65	2.33	21.35	19.48	12.04
150 items 2% cluster size	69	44	15	0.11	0.34	0.45	4.19	3.75	4.34	4.30	4.10	4.79
150 items 3% cluster size	70	27	13	1.31	1.00	0.77	4.06	3.07	3.82	5.37	4.08	4.59
150 items 5% cluster size	69	18	12	3.79	2.22	1.66	3.90	2.32	3.23	7.70	4.54	4.89
150 items 7% cluster size	70	13	11	4.98	4.04	3.11	3.70	2.21	3.06	8.68	6.25	6.17
150 items 10% cluster size	70	9	10	7.54	7.13	4.17	3.75	2.48	2.35	11.29	9.61	6.52
150 items 15% cluster size	69	6	8	12.41	12.04	7.87	3.39	2.61	2.30	15.80	14.65	10.18
150 items 20% cluster size	70	4	6	17.74	16.45	12.36	3.29	1.73	3.07	21.04	18.18	15.43
200 items 2% cluster size	123	59	20	0.96	0.59	0.42	4.42	4.05	4.41	5.38	4.65	4.83
200 items 3% cluster size	122	41	18	1.90	1.07	0.81	4.38	3.55	4.02	6.29	4.62	4.83
200 items 5% cluster size	122	23	16	3.77	2.70	1.63	4.22	3.15	3.53	7.99	5.86	5.15
200 items 7% cluster size	122	17	15	5.60	5.24	2.63	4.14	3.30	2.66	9.73	8.55	5.29
200 items 10% cluster size	122	12	14	8.37	7.68	4.91	3.79	3.11	2.76	12.17	10.79	7.66
200 items 15% cluster size	121	8	10	12.64	11.99	8.47	3.54	2.89	2.56	16.19	14.89	11.04
200 items 20% cluster size	121	6	9	17.85	17.36	11.34	3.23	2.73	2.63	21.08	20.08	13.97
400 items 2% cluster size	488	118	42	1.34	0.49	0.34	4.52	3.67	4.08	5.86	4.16	4.42
400 items 3% cluster size	490	81	40	2.19	0.74	0.54	4.39	2.94	3.60	6.57	3.68	4.14
400 items 5% cluster size	491	46	38	4.05	2.21	1.57	4.22	2.38	3.25	8.27	4.58	4.81
400 items 7% cluster size	488	32	35	5.87	4.50	2.57	4.14	2.50	2.53	10.01	7.00	5.09
400 items 10% cluster size	486	23	28	8.74	7.39	4.35	3.90	2.55	2.25	12.63	9.94	6.60
400 items 15% cluster size	486	16	19	12.95	11.68	9.42	3.60	2.33	3.42	16.54	14.00	12.85
400 items 20% cluster size	486	11	18	18.08	17.40	11.63	3.22	2.54	2.66	21.30	19.94	14.29

Table B-5: Results on "Article" data sets for all-pair-comparison-based (AC), selected-comparison-based (SC), and M-tree-base (MT) heuristics with uniform weights using terms that are separated by no more than *seven* variations in the spelling

Test Description	Time (sec)			False alarm (%)			False dismissal (%)			Error (%)		
	AC	SC	MT	AC	SC	MT	AC	SC	MT	AC	SC	MT
50 items 3% cluster size	9	8	5	0.82	1.12	0.86	2.78	3.08	3.49	3.59	4.20	4.35
50 items 5% cluster size	9	5	4	0.82	2.78	1.76	2.78	2.78	3.22	3.59	5.55	4.98
50 items 7% cluster size	9	4	4	4.73	4.71	2.82	2.78	2.76	3.00	7.51	7.47	5.82
50 items 10% cluster size	9	3	3	4.73	6.71	4.00	2.78	2.55	2.61	7.51	9.27	6.61
50 items 15% cluster size	9	2	3	12.41	12.14	7.59	2.61	2.35	2.29	15.02	14.49	9.88
50 items 20% cluster size	9	1	3	16.82	16.08	9.96	2.45	1.71	1.45	19.27	17.80	11.41
100 items 2% cluster size	35	34	13	0.18	0.37	0.44	3.98	4.13	4.55	4.16	4.51	4.99
100 items 3% cluster size	35	23	10	0.20	0.98	0.73	3.96	3.75	4.19	4.16	4.73	4.92
100 items 5% cluster size	35	13	8	2.10	2.40	1.58	3.84	3.13	3.58	5.94	5.54	5.16
100 items 7% cluster size	35	9	7	5.66	4.87	2.92	3.52	2.73	2.62	9.17	7.61	5.54
100 items 10% cluster size	35	7	7	7.58	7.22	4.91	3.43	2.90	3.22	11.01	10.12	8.13
100 items 15% cluster size	35	5	7	11.45	11.63	6.94	3.33	2.76	2.35	14.79	14.39	9.30
100 items 20% cluster size	35	3	5	17.17	17.10	12.35	3.13	2.68	2.98	20.30	19.78	15.34
150 items 2% cluster size	79	52	17	0.18	0.68	0.50	4.43	4.27	4.58	4.61	4.96	5.08
150 items 3% cluster size	79	32	16	1.41	1.47	1.00	4.32	3.72	4.10	5.74	5.19	5.10
150 items 5% cluster size	79	20	12	3.64	3.09	1.87	3.93	3.37	3.50	7.57	6.46	5.37
150 items 7% cluster size	79	14	12	5.09	4.72	3.41	3.98	3.06	3.39	9.07	7.79	6.80
150 items 10% cluster size	79	11	11	7.58	7.43	4.74	3.96	2.97	2.69	11.54	10.40	7.42
150 items 15% cluster size	79	7	9	12.65	12.13	8.32	3.81	2.89	2.52	16.47	15.02	10.84
150 items 20% cluster size	79	6	7	17.79	17.49	12.15	3.52	2.96	2.58	21.31	20.45	14.73
200 items 2% cluster size	140	68	23	0.98	0.53	0.48	4.41	3.95	4.44	5.39	4.48	4.92
200 items 3% cluster size	140	46	21	1.80	1.01	0.60	4.25	3.44	3.86	6.05	4.45	4.46
200 items 5% cluster size	140	28	20	3.65	2.64	1.38	4.10	3.05	3.35	7.74	5.69	4.73
200 items 7% cluster size	139	20	19	5.48	4.67	2.36	3.97	2.68	2.42	9.46	7.35	4.78
200 items 10% cluster size	140	13	18	8.25	7.67	3.89	3.72	3.05	2.16	11.97	10.72	6.05
200 items 15% cluster size	139	9	13	12.63	11.66	7.38	3.48	2.51	1.94	16.11	14.17	9.32
200 items 20% cluster size	140	6	11	17.86	17.01	9.68	3.38	2.33	1.47	21.24	19.34	11.16
400 items 2% cluster size	539	133	43	1.43	0.65	0.53	4.71	3.93	4.34	6.14	4.58	4.87
400 items 3% cluster size	537	92	41	2.34	1.20	0.92	4.64	3.50	3.98	6.98	4.71	4.90
400 items 5% cluster size	536	54	39	4.22	2.57	2.13	4.50	2.84	3.76	8.72	5.40	5.89
400 items 7% cluster size	533	36	38	6.08	4.49	3.67	4.45	2.59	3.63	10.53	7.08	7.29
400 items 10% cluster size	536	27	29	9.00	7.39	5.20	4.26	2.65	3.11	13.27	10.04	8.31
400 items 15% cluster size	539	18	21	13.22	11.70	9.40	3.97	2.45	3.32	17.19	14.15	12.72
400 items 20% cluster size	534	12	20	18.42	17.29	11.58	3.66	2.52	2.86	22.08	19.81	14.44

Table B-6: Results on "PhdThesis" data sets for all-pair-comparison-based (AC), selected-comparison-based (SC), and M-tree-base (MT) heuristics with uniform weights using terms that are separated by no more than *two* variations in the spelling

Test Description	Time (sec)			False alarm (%)			False dismissal (%)			Error (%)		
	AC	SC	MT	AC	SC	MT	AC	SC	MT	AC	SC	MT
50 items 3% cluster size	4	4	3	0.24	0.76	0.63	2.29	2.71	3.45	2.53	3.47	4.08
50 items 5% cluster size	4	2	2	0.33	2.31	1.49	2.29	2.31	3.12	2.61	4.61	4.61
50 items 7% cluster size	4	2	2	4.24	3.88	2.12	2.29	1.92	2.49	6.53	5.80	4.61
50 items 10% cluster size	4	1	2	4.24	6.35	2.98	2.29	2.18	2.18	6.53	8.53	5.16
50 items 15% cluster size	4	1	2	12.08	11.55	6.76	2.29	1.76	1.57	14.37	13.31	8.33
50 items 20% cluster size	4	1	1	15.92	15.73	9.02	2.29	1.37	1.51	18.20	17.10	10.53
100 items 2% cluster size	14	14	6	0.14	0.25	0.34	3.92	4.03	4.52	4.06	4.27	4.86
100 items 3% cluster size	14	9	5	0.14	0.63	0.59	3.92	3.41	4.11	4.06	4.04	4.70
100 items 5% cluster size	14	5	4	1.76	1.93	1.15	3.52	2.68	3.31	5.27	4.62	4.46
100 items 7% cluster size	14	4	3	5.64	4.35	2.35	3.52	2.23	2.53	9.15	6.59	4.88
100 items 10% cluster size	14	3	3	7.58	6.56	3.58	3.27	2.25	2.01	10.85	8.81	5.59
100 items 15% cluster size	14	2	3	11.29	11.20	6.13	3.19	2.35	1.73	14.48	13.55	7.85
100 items 20% cluster size	14	1	2	17.56	16.47	10.42	3.15	2.07	1.39	20.71	18.55	11.81
150 items 2% cluster size	34	22	8	0.04	0.32	0.31	4.13	3.74	4.24	4.18	4.07	4.55
150 items 3% cluster size	34	14	8	1.33	0.84	0.81	4.10	2.91	3.79	5.43	3.75	4.61
150 items 5% cluster size	34	8	6	3.91	2.21	1.15	4.03	2.33	2.81	7.94	4.55	3.96
150 items 7% cluster size	34	6	6	5.27	3.75	2.02	3.99	1.93	2.06	9.26	5.68	4.09
150 items 10% cluster size	34	4	6	7.63	6.68	3.27	3.85	2.04	1.58	11.48	8.72	4.85
150 items 15% cluster size	34	3	4	12.55	11.59	6.81	3.54	2.17	1.43	16.10	13.76	8.25
150 items 20% cluster size	34	2	3	17.54	16.36	9.92	3.10	1.66	1.33	20.64	18.02	11.25
200 items 2% cluster size	62	30	11	0.88	0.28	0.32	4.25	3.63	4.22	5.14	3.91	4.54
200 items 3% cluster size	62	21	10	1.84	0.76	0.52	4.21	3.13	3.73	6.06	3.89	4.25
200 items 5% cluster size	62	13	9	3.85	1.76	1.16	4.23	2.10	2.81	8.08	3.86	3.97
200 items 7% cluster size	62	8	8	5.62	4.73	2.12	4.05	2.68	2.25	9.67	7.42	4.37
200 items 10% cluster size	62	6	8	8.52	6.64	3.42	3.93	1.95	1.65	12.45	8.60	5.07
200 items 15% cluster size	63	4	6	12.92	11.63	7.30	3.71	2.42	1.54	16.63	14.05	8.84
200 items 20% cluster size	63	3	5	17.94	16.04	9.90	3.40	1.31	1.13	21.34	17.35	11.03
300 items 2% cluster size	123	39	16	1.13	0.36	0.25	4.44	3.66	4.13	5.58	4.02	4.38
300 items 3% cluster size	124	28	14	1.64	0.68	0.58	4.30	3.00	3.69	5.94	3.68	4.27
300 items 5% cluster size	123	16	13	3.52	2.08	1.04	4.20	2.37	2.68	7.72	4.45	3.72
300 items 7% cluster size	121	11	13	5.87	4.22	2.14	3.95	2.30	2.50	9.81	6.52	4.65
300 items 10% cluster size	121	8	10	8.53	6.73	3.79	3.81	2.01	1.91	12.35	8.75	5.71
300 items 15% cluster size	122	6	8	12.54	11.41	6.94	3.46	2.18	2.03	16.00	13.59	8.97
300 items 20% cluster size	121	4	7	17.92	16.78	9.60	3.17	2.03	1.87	21.09	18.80	11.46
400 items 2% cluster size	233	57	23	1.24	0.40	0.30	4.45	3.61	4.09	5.69	4.01	4.39
400 items 3% cluster size	232	39	22	2.14	0.83	0.51	4.36	3.05	3.60	6.50	3.88	4.11
400 items 5% cluster size	232	23	20	4.04	1.77	1.25	4.27	1.97	2.66	8.31	3.73	3.91
400 items 7% cluster size	232	17	17	5.86	4.45	2.21	4.15	2.48	1.99	10.01	6.93	4.20
400 items 10% cluster size	232	12	14	8.71	6.71	3.46	3.95	1.90	1.58	12.66	8.61	5.04
400 items 15% cluster size	233	8	11	12.90	11.32	6.97	3.58	2.00	1.93	16.48	13.32	8.90
400 items 20% cluster size	234	5	9	18.07	16.49	10.71	3.33	1.65	1.15	21.40	18.14	11.86

LIST OF REFERENCES

[1] S. Abiteboul, "Querying semistructured data," *International Conference on Database Theory*, vol. 6, pp. 1-18, 1997.

[2] Y. Arens, C. Y. Chee, C. Hsu, and C. Knoblock, "Retrieving and Integrating Data from Multiple Information Sources," *International Journal of Intelligent & Cooperative Information Systems*, vol. 2, pp. 127-158, 1993.

[3] AT&T, "XML-QL: A Query Language for XML, Version 0.9," available at http://www.research.att.com/~mff/xmlql/, 2000.

[4] P. Atzeni, G. Mecca, and P. Merialdo, "To Weave the Web," *Proceedings of 23rd International Conference on Very Large Data Bases*, Athens, Greece, 1997.

[5] C. Baru, A. Gupta, B. Ludascher, R. Marciano, Y. Papakonstantinou, and P. Velikhov, "XML-Based Information Mediation with MIX," *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, Philadelphia, Pennsylvania, 1999.

[6] C. Batini, M. Lenzerini, and S. B. Navathe, "A Comparative Analysis of Methodologies for Database Schema Integration," *ACM Computing Surveys*, vol. 18, pp. 323-364, 1986.

[7] N. Beckmann, H. P. Kriegel, R. Schneider, and B. Seeger, "The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles," *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, Atlantic City, New Jersey, 1990.

[8] S. Berchtold, D. A. Keim, and H.-P. Kriegel, "The X-tree: An Index Structure for High-Dimensional Data," *Proceedings of 22nd International Conference on Very Large Data Bases*, Mumbai (Bombay), India, 1996.

[9] S. Bergamaschi, S. Castano, and M. Vincini, "Semantic Integration of Semistructured and Structured Data Sources," *SIGMOD Record*, vol. 28, pp. 54-59, 1999.

[10] T. Bozkaya and M. Ozsoyoglu, "Distance-Based Indexing For High-Dimensional Metric Spaces," *Proceedings ACM SIGMOD International Conference on Management of Data*, Tucson, Arizona, 1997.

[11] S. Bressan, C. H. Goh, K. Fynn, M. Jakobisiak, K. Hussein, H. B. Kon, T. Lee, S. E. Madnick, T. Pena, J. Qu, A. W. Shum, and M. Siegel, "The COntext INterchange Mediator Prototype," *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, Tucson, Arizona, 1997.

[12] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom, "The TSIMMIS Project: Integration of Heterogeneous Information Sources," *The 10th Anniversary Meeting of the Information Processing Society of Japan*, Tokyo, Japan, 1994.

[13] P. Ciaccia, M. Patella, and P. Zezula, "M-tree: An Efficient Access Method for Similarity Search in Metric Spaces," *Proceedings of 23rd International Conference on Very Large Data Bases*, Athens, Greece, 1997.

[14] S. Cluet, C. Delobel, J. Simeon, and K. Smaga, "Your Mediators Need Data Conversion!," *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, Seattle, Washington, 1998.

[15] S. Cluet, S. Jacqmin, and J. Simeon, "The New YaTL: Design and Specification," Technical Report, The French National Institute for Research in Computer Science and Control (INRIA), Paris, France, 1999.

[16] W. W. Cohen, "The WHIRL Approach to Data Integration," *IEEE Intelligent Systems*, vol. 13, pp. 20-24, 1998.

[17] CommerceOne, http://www.commerceone.com/, 1999.

[18] U. Dayal and H. Hwang, "View Definition and Generalization for Database Integration in Multibase: A System for Heterogeneous Distributed Databases," *TOSE*, vol. 10, pp. 628--644, 1984.

[19] A. Deutch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu, "A Query Language for XML," *Proceedings of the 8th International World Wide Web Conference (WWW8)*, Toronto, Canada, 1999.

[20] D. Dey, S. Sarkar, and P. De, "Entity Matching in Heterogeneous Databases: A Distance-Based Decision Model," *Proceedings of the 31st Annual Hawaii International Conference on System Sciences*, Kohala Coast, Hawaii, 1999.

[21] D. Dey, S. Sarkar, and P. De, "A Probabilistic Decision Model for Entity Matching in Heterogeneous Databases," *Management Science*, vol. 44, pp. 1379-1395, 1998.

[22] D. Dreilinger and A. E. Howe, “Experiences with Selecting Search Engines Using Metasearch,” *ACM Transaction on Information Systems*, vol. 15, pp. 195-222, 1997.

[23] O. M. Duschka and M. R. Genesereth, “Infomaster -- An Information Integration Tool,” *Proceedings of the International Workshop Intelligent Information Integration during the 21st German Annual Conference on Artificial Intelligence (KI-97)*, Freiburg, Germany, 1997.

[24] C. Faloutsos, *Searching Multimedia Databases by Content*. Boston: Kluwer Academic Publishers, 1996.

[25] C. Faloutsos and I. Kamel, “Packed R-trees Using Fractals,” Technical Report CS-TR-3009, University of Maryland, College Park, December 1993.

[26] A. Farquhar, R. Fikes, and J. Rice, “The ontolingua server: A tool for collaborative ontology construction,” Technical report KSL-96-26, Knowledge System Laboratory, Stanford University, Stanford, CA, September 1996.

[27] M. Fernandez, D. Florescu, J. Kang, A. Levy, and D. Suciu, “STRUDEL: A Web-site Management System,” *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, Tucson, Arizona, 1997.

[28] M. Fernandez, J. Simeon, P. Wadler, S. Cluet, A. Deutsch, D. Florescu, A. Levy, D. Maier, J. McHugh, J. Robie, D. Suciu, and J. Widom, “XML Query Languages: Experiences and Exemplars,” The World Wide Web Consortium (W3C), http://www.w3.org/1999/09/ql/docs/xquery.html September 1999.

[29] M. R. Genesereth, A. M. Keller, and O. M. Duschka, “Infomaster: An Information Integration System,” *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, Tucson, Arizona, 1997.

[30] R. Goldman, J. McHugh, and J. Widom, “From Semistructured Data to XML: Migrating the Lore Data Model and Query Language,” *Proceedings of the 1999 ACM SIGMOD Workshop on the Web and Databases (WebDB'99)*, Philadelphia, Pennsylvania, 1999.

[31] T. R. Gruber, “A Translation Approach to Portable Ontologies,” *Knowledge Acquisition*, vol. 5, pp. 199-220, 1993.

[32] A. Guttman, “R-Trees: A Dynamic Index Structure for Spatial Searching,” *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, Boston, Massachusetts, 1984.

[33] L. M. Haas, D. Kossman, E. L. Wimmers, and J. Yang, "Optimizing Queries Across Diverse Data Sources," *Proceedings of the 23rd International Conference on Very Large Data Bases*, Athens, Greece, 1997.

[34] J. Hammer, "The Information Integration Wizard (IWiz) Project," Technical Report TR99-019, University of Florida, Gainesville, FL, October 1999.

[35] J. Hammer, H. Garcia-Molina, J. Widom, W. Labio, and Y. Zhuge, "The Stanford Data Warehousing Project," *Data Engineering Bulletin*, vol. 18, pp. 41-48, 1995.

[36] E. R. Harold, *XML Bible*, Foster, California: IDG Books Worldwide, 1999.

[37] E. R. Harold, *XML: Extensible Markup Language*, Foster, California: IDG Books Worldwide, 1998.

[38] I. Kamel and C. Faloutsos, "Hilbert R-trees: An improved R-tree using fractals," Technical Report CS-TR-3032, Univ. of Maryland, College Park, February 1993.

[39] V. Kashyap and A. Sheth, "Semantic and Schematic Similarities between Objects in Databases: A Context-based approach," Technical Report, Department of Computer Science, University of Georgia, Atlanta, Georgia, February 16 1995.

[40] W. Kent, "The Many Forms of a Single Fact," Technical Report, HP Labs, Palo Alto, CA, June 1988.

[41] W. Kent, "Solving Domain Mismatch and Schema Mismatch Problems with an Object-Oriented Database Programming Language," *Proceedings of the 17th International Conference on Very Large Data Bases*, Barcelona, Spain, 1991.

[42] W. Kim, I. Choi, S. Gala, and M. Scheevel, "On Resolving Schematic Heterogeneity in Multidatabase Systems," *Distributed and Parallel Databases*, Boston: Kluwer Academic Publishers, pp. 251-279, 1993.

[43] KSL, "Ontology Server," Knowledge Systems Laboratory: http://www-ksl-svc.stanford.edu:5915, 1992.

[44] KSL, "Stanford Knowledge Sharing Group," Knowledge Systems Laboratory: http://www-ksl.stanford.edu/, 1998.

[45] H. W. Kuhn, "The Hungarian Method for the Assignment Algorithm," *Naval Research Logistics Quarterly*, vol. 1, pp. 83-97, 1955, March-June.

[46] W. Labio, Y. Zhuge, J. L. Wiener, H. Gupta, H. Garcia-Molina, and J. Widom, "The WHIPS Prototype for Data Warehouse Creation and Maintenance," , *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, Tucson, Arizona, 1997.

[47] L. V. S. Lakshmanan, F. Sadri, and I. N. Subramanian, “SchemaSQL - A Language for Interoperability in Relational Multi-Database Systems,” *Proceedings of the 22nd International Conference on Very Large Data Bases*, 1996.

[48] E. L. Lawler, *Combinatorial Optimization: Networks and Matroids*. New York: Holt, Rinehart and Winston, 1976.

[49] A. Levy, “The Information Manifold Approach to Data Integration,” *IEEE Intelligent Systems*, vol. 13, pp. 12 - 16, 1998.

[50] K.-I. Lin, H. V. Jagadish, and C. Faloutsos, “The TV-Tree: An Index Structure for High-Dimensional Data,” *VLDB Journal*, vol. 3, pp. 517-542, 1994.

[51] D. A. Maluf and G. Wiederhold, “Abstraction of Representation for Interoperation,” *The 10th International Symposium on Methodologies for Intelligent Systems (ISMIS)*, Charlotte, North Carolina, 1997.

[52] R. J. Miller, “Using Schematically Heterogeneous Structures,” *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, Seattle, Washington, 1998.

[53] R. S. Oliveira, “Resolving Structural Conflicts during Integration of XML Data Sources,” Master's Thesis, Computer and Information Science and Engineering Department, University of Florida, Gainesville, Florida, 1999.

[54] Oracle, “Oracle XML Developer's Kit for Java,” available at http://technet.oracle.com/tech/xml/parser_java2/index.htm, October 2000.

[55] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*. Englewood, New Jersey: Prentice-Hall, 1982.

[56] C. Pluempitiwiriyawej and J. Hammer, “A Classification Scheme for Semantic and Schematic Heterogeneities in XML Data Sources,” Technical Report 00-004, Computer and Information Science and Engineer, University of Florida, Gainesville, September 2000.

[57] M. Roantree, J. Murphy, and W. Hasselbring, “The OASIS Multidatabase Prototype,” *SIGMOD Record*, vol. 28, pp. 97-103, 1999.

[58] RosettaNet, http://www.rosettanet.org, 1999.

[59] SchemaNet, http://www.schema.net, 1999.

[60] E. Sciore, M. Siegel, and A. Rosenthal, “Using Semantic Values to Facilitate Interoperability Among Heterogeneous Information Systems,” *The ACM Transactions on Database Systems (TODS)*, vol. 19, pp. 254-290, 1994.

[61] T. K. Sellis, N. Roussopoulos, and C. Faloutsos, “The R+Tree: A Dynamic Index for Multi-Dimensional Objects,” *Proceedings of 13th International Conference on Very Large Data Bases*, Brighton, England, 1987.

[62] A. Shah, “Source Specific Query Rewriting and Query Plan Generation for Merging XML-Based Semistructured data in Mediation Systems,” Master's Thesis, Computer and Information Science and Engineering Department, University of Florida, Gainesville, Florida, 2001.

[63] M. Siegel and S. E. Madnick, “A Metadata Approach to Resolving Semantic Conflicts,” *The 17th International Conference on Very Large Databases*, Barcelona, Spain, 1991.

[64] A. Teterovskaya, “Conflict Detection and Resolution During Restructuring of XML Data,” Master's Thesis, Computer and Information Science and Engineering Department, University of Florida, Gainesville, Florida 2000.

[65] J. K. Uhlmann, “Satisfying General Proximity/Similarity Queries with Metric Trees,” *Information Processing Letter*, vol. 40, pp. 175-179, 1991.

[66] W3C, “The Document Object Model (DOM) Level 1 Specification,” The World Wide Web Consortium, http://www.w3.org/TR/REC-DOM-Level-1/, October 1998.

[67] W3C, “The Document Object Model (DOM) Level 2 Core Specification,” The World Wide Web Consortium, http://www.w3.org/TR/DOM-Level-2-Core/, November 2000.

[68] W3C, “Document Object Model (DOM) Level 2 Events Specification,” The World Wide Web Consortium, http://www.w3.org/TR/DOM-Level-2-Events/, November, 2000.

[69] W3C, “Document Object Model (DOM) Level 2 Style Specification,” The World Wide Web Consortium, http://www.w3.org/TR/DOM-Level-2-Style/, November 2000.

[70] W3C, “Document Object Model (DOM) Level 2 Views Specification,” The World Wide Web Consortium, http://www.w3.org/TR/DOM-Level-2-Views/, November 2000.

[71] W3C, “The Document Object Model (DOM) Level 3 Core Specification,” The World Wide Web Consortium, http://www.w3.org/TR/DOM-Level-3-Core/, January 2001.

[72] W3C, “Extensible Markup Language (XML) 1.0,” The World Wide Web Consortium, http://www.w3c.org/TR/1998/REC-xml-19980210.html, February 1998.

[73] W3C, “Extensible Stylesheet Language (XSL),” The World Wide Web Consortium, http://www.w3c.org/TR/xsl/, November 1998.

[74] W3C, “Namespaces in XML,” http://www.w3.org/TR/REC-xml-names/, The World Wide Web Consortium, 1999.

[75] W3C, “XML Linking Language (XLink),” http://www.w3.org/TR/xlink/: The World Wide Web Consortium, 2000.

[76] W3C, “XML Pointer Language (XPointer),” http://www.w3.org/TR/xptr: The World Wide Web Consortium, 1999.

[77] W3C, “The XML Query Algebra,” The World Wide Web Consortium, http://www.w3.org/TR/query-algebra/, December 2000.

[78] W3C, “XML Schema Part 0: Primer,” The World Wide Web Consortium, http://www.w3.org/TR/xmlschema-0/, February 2000.

[79] W3C, “XML Schema Part 1: Structures,” The World Wide Web Consortium, http://www.w3.org/TR/xmlschema-1/, February 2000.

[80] W3C, “XML Schema Part 2: Datatypes,” The World Wide Web Consortium, http://www.w3.org/TR/xmlschema-2/, February 2000.

[81] W3C, “XSL Transformation (XSLT),” The World Wide Web Consortium, http://www.w3.org/TR/xslt, November1999.

[82] G. Wiederhold, “Scalable Knowledge Composition (SKC),” 1997.

[83] XML/EDI, http://www.xmledi.com, 1999.

[84] C. Yu and W. Meng, “Metasearch Engine,” in *Encyclopedia of Distributed Computing*, P. Dasgupta and J. Urban, Eds.: Kluwer Academic Publishers, 2000.

[85] C. Yu and W. Meng, “Metasearch Engine: Solutions and Challenges,” *International Conference on Very Large Data Bases Tutorial*, Edinburgh, UK, 1999.

[86] G. Zhou, R. Hull, R. King, and J.-C. Franchitti, “Data Integration and Warehousing Using H2O,” *Data Engineering Bulletin*, vol. 18, pp. 29-40, 1995.

[87] J. Zobel and A. Moffat, “Exploring the Similarity Space,” *SIGIR Forum*, vol. 32, pp. 18-34, 1998.

BIOGRAPHICAL SKETCH

Charnyote Pluempitiwiriyawej was born in Bangkok, Thailand. He received a BEng degree in computer engineering from the King Mongkut's Institution of Technology Thonburi (Thailand) in 1994; an MS degree in computer science from the University of Maryland, College Park, in 1997; and a Ph.D. degree from the Department of Computer and Information Science and Engineering, University of Florida, in 2001. His research interests include XML-based semistructured data integration and management, data warehousing and mediation systems, and database and World Wide Web technologies.