A HOLISTIC PARADIGM FOR LARGE SCALE SCHEMA MATCHING

BY

BIN HE

B.S., Peking University, 1998
M.S. Peking University, 2000
M.S., University of Illinois, 2002

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2006

Urbana, Illinois

Abstract

Schema matching is a critical problem for integrating heterogeneous information sources. Traditionally, the problem of matching multiple schemas has essentially relied on finding pairwise attribute correspondences in isolation. In contrast, this thesis proposes a new matching paradigm, *holistic schema matching*, to match many schemas at the same time and find all matchings at once. By handling a set of schemas together, we can explore their *context* information that reflects the semantic correspondences among attributes. Such information is not available when schemas are matched only in pairs. As the realizations of holistic schema matching, we develop two approaches in sequence. To begin with, we develop the MGS framework, which finds simple 1:1 matchings by viewing schema matching as hidden model discovery. Then, to deal with complex matchings, we further develop the DCM framework by abstracting schema matching as correlation mining. Further, to automate the entire matching process, we incorporate the DCM framework with automatically extracted interfaces and find that the inevitable errors in automatic interface extraction may significantly affect the matching result. To make the DCM framework robust against such "noisy" schemas, we propose to integrate it with an *ensemble* approach by randomizing the schema data into multiple DCM matchers and aggregating their ranked results by taking majority voting. Last, as our matching algorithms require large scale schemas in the same domain (*e.g.*, Books and Airfares) as input, we develop an *object-focused* crawler for effectively collecting query interfaces and a *model-differentiation* based clustering approach to clustering schemas into their domain hierarchy.

Table of Contents

Chapter

List of Figures

Chapter 1

Introduction

Schema matching is fundamental for enabling query mediation and data exchange across information sources [6, 61]. This thesis proposes a new matching paradigm, *holistic schema matching*, which is realized by two approaches we developed recently with global and local evaluation strategies respectively. Traditionally, schema matching has been approached mainly by finding *pairwise attribute correspondences*, to construct an integrated schema for two (or some small number of n) sources. We observe that there are often challenges (and certainly also opportunities) to deal with large numbers of sources. In such scenarios, the challenge of large scale can itself be an opportunity for new approaches– We can take a holistic view of all the input schemas and find all the matchings at once.

Such scenarios arise, in particular, for integrating databases on the Internet, or the so-called "deep Web." A July 2000 survey [7] estimated that 96,000 "search cites" and 550 billion content pages on this deep Web. Our recent study [16] in April 2004 estimated 450,000 online databases. With the virtually unlimited amount of information, the deep Web is clearly an important frontier for data integration. On this deep Web, numerous online databases provide data via their *query interfaces*, instead of static URL links. Each query interface accepts queries over its *query schemas* (*e.g.*, author, title, subject, ... for

amazon.com). *Schema matching*, *i.e.*, discovering semantic correspondences of attributes, across Web interfaces is essential for mediating queries across deep Web sources.

Matching Web interfaces in the same domain (*e.g.*, Books, Airfares) is a particularly important problem with broad applications. We often need to search over alternative sources in the same domain such as purchasing a book (or flight ticket) across many online book (or airline) sources. Given a set of Web interfaces in the same domain, this thesis solves the problem of discovering matchings among those interfaces. In particular, our MetaQuerier project (http://metaquerier.cs.uiuc.edu) is aiming at developing techniques to automatically build domain portals [18]. The work presented by this thesis is a critical component of the MetaQuerier project, *i.e.*, the schema matching subsystem of the MetaQuerier.

However, existing schema matching work mostly focuses on small scale integration by finding pairwise attribute correspondences between two sources. Traditionally, schema matching relies on matchings between pairwise attributes before integrating multiple schemas. For instance, traditional binary or *n*-ary [55] schema integration methodologies (as [6] surveys) exploit pairwise attribute correspondence assertions (mostly manually given) for merging two or some *n* sources. Further, recent work on automatic schema matching mostly focuses on matchings between two schemas (*e.g.*, [28, 50, 52, 47]). Based on this fact, the latest survey [60] abstracts schema matching as pairwise similarity mappings between two input sources.

To tackle the challenge of large scale matching, as well as to take advantage of its new opportunity, we propose a new paradigm, *holistic schema matching*, to match many schemas at the same time and find all the matchings at once, as Figure 1.1 shows. In particular, holistic schema matching takes a set of schemas as input and outputs a semantic *model*, which contains all the matchings among the input schemas (*e.g.*, a model of book schemas may contain author = writer = name, subject = category, ...). Such a holistic view enables us to explore the *context* information beyond two schemas (*e.g.*, similar

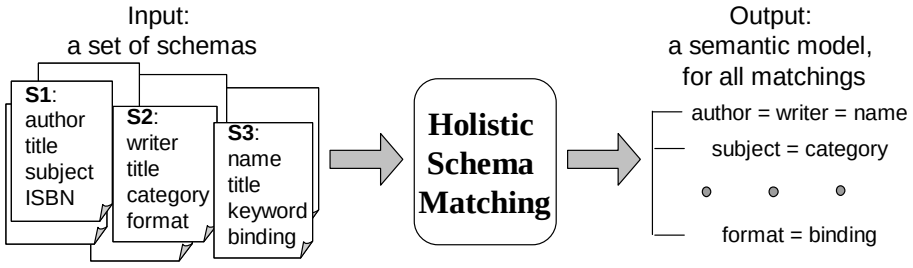


Figure 1.1: The holistic schema matching paradigm.

attributes across multiple schemas; co-occurrence patterns among attributes), which is not available when schemas are matched only in pairs.

Compared with traditional approaches, we believe the holistic approach has several advantages: First, *scalability*: By unifying a large number of input schemas holistically rather than matching attributes pairwise, it addresses the scale of matching required in the new frontier of networked databases, such as our motivating goal of the deep Web. Second, *solvability*: In fact, the large scale can itself be a crucial leverage to make schema matching more solvable– in particular, it enables effective exploration of the context information. Such context information will be more sufficient as more sources are exploited. Intuitively, we are building upon the "peer context" among schemas. Being context-based, the holistic matching will benefit from the scale: the accuracy will "scale" with the number of sources. For instance, our specific MGS and DCM approaches, as we will discuss, are both statistical methods, which will thus benefit from more "observations."

With the holistic paradigm, this thesis proposes two approaches we developed in sequence as its realizations. To begin with, we develop the MGS framework [37] with a *global evaluation* strategy to deal with simple 1:1 matchings (*i.e.*, matching between two attributes such as author = writer). Global evaluation exhaustively evaluates all possible models and selects the best one among them. The best model contains the set of matchings with the highest overall confidence to assemble the correct model.

In particular, the MGS framework [37] realizes such global evaluation by hypothesizing the existence of a hidden generative model for each domain (*e.g.*, Books, Movies) (Chapter 2). Under this hypothesis, a schema can be viewed as an instance generated from the model with some probabilistic behavior. Schema matching is thus transformed into the discovery of the hidden model, given a set of schema instances. To realize such hidden model discovery, we develop the MGS framework, which discovers matchings with statistical hypothesis testing.

While the MGS framework can effectively model simple matchings, it cannot find complex matchings, which generally exist across Web query interfaces (*e.g.*, author is a synonym of the grouping of last name and first name in Books domain, *i.e.*, author = {last name, first name}). To discover complex matchings, we further develop the DCM framework [39] with a *local evaluation* strategy. Local evaluation independently assesses every single matching and then incrementally constructs the model. Instead of exhaustively enumerating all the possible models, local evaluation approximately searches for the best model by constructing it incrementally. For instance, among all the potential matchings in book schemas, we may first select the most confident matching subject = category and consider it as part of the best model. Then we iteratively select the next most confident matching under this partial model result, toward eventually completing the best model.

In particular, the DCM framework [39] realizes such local evaluation based on the observation that co-occurrence patterns across schemas often reveal the complex relationships of attributes (Chapter 3). Specifically, we observe that *grouping attributes* (*e.g.*, {first name, last name}) tend to be co-present in query interfaces and thus positively correlated. In contrast, *synonym attributes* are negatively correlated because they rarely co-occur. This insight motivates us to develop the DCM framework, which greedily discovers complex matchings with a dual mining of positive and negative correlations.

Further, to complete an automatic matching process, which starts from raw HTML pages, we integrate the DCM framework with an automatic interface extractor [72]. Such "system integration" turns out to be non-trivial– As automatic interface extraction cannot be perfect, it will introduce "noise" (*i.e.*, erroneous extraction), which challenges the performance of the subsequent matching algorithm. As Chapter 4 will discuss, the errors in the interface extraction step may affect the correlations of matchings and consequently the matching result.

To make the DCM framework robust against noise, we integrate it with an *ensemble* scheme, which aggregates a multitude of the DCM matchers to achieve robustness, by exploiting statistical sampling and majority voting (Chapter 4). Specifically, we randomly sample a subset of schemas (as a *trial*) to match, instead of using all the schemas. Intuitively, it is likely that such a trial still contains sufficient attribute information to match while removing certain noisy schemas. Further, we conduct multiple independent trials. Since errors in different trials are independent, when noise is relatively few, it is likely that only a minority of trials are affected. We thus take majority voting among the discovered matching of all trials to achieve the robustness of holistic matching.

Last, since our matching algorithms require the input schemas (*i.e.*, query interfaces) from the same domain, to enable such large scale matching, we need to develop automatic techniques to discover query interfaces on the Web (*i.e.*, the source discovery problem) and cluster them into their domain hierarchy (*i.e.*, the schema clustering problem).

For the source discover problem, we develop a *Web Form Crawler* to collect query interfaces (*i.e.*, query forms) across various domains in both efficient and comprehensive manners (Chapter 5). In particular, query forms, while many, when compared with the size of the Web, are sparsely scattered among pages, which brings new challenges for crawling: First, due to the topic-neutral nature of our crawling problem, we cannot rely on existing topic-focused crawling techniques. Second, traditional page-based

crawling techniques cannot achieve a good balance between crawling harvest and coverage. As a new attempt, we propose a *structure-driven* crawling framework by observing *structure locality* of query forms– That is, query forms are often close to root pages of Web sites and accessible by following navigational links. Exploring this structure locality, we substantiate the structure-driven crawling framework into a *site-based* Web Form Crawler by first collecting the site entrances, as the Site Finder, and then searching for query forms within the scope of each site, as the Form Finder.

For the schema clustering problem, by viewing schemas as a type of categorical data, we translate the problem into the clustering of categorical data and develop a *model-differentiation* based clustering approach (Chapter 6). Specifically, our approach pursues probabilistic model-based clustering with a new objective function. To begin with, motivated by our real-world observations, we hypothesize that homogeneous sources share the same hidden generative model, which probabilistically generates schemas from a finite vocabulary of attributes. This hypothesis naturally matches model-based clustering– to form clusters from different models. Further, to realize such clustering, we propose a new objective function: *model-differentiation* or MD, which seeks to maximize *statistical heterogeneity* among clusters. Rather than relying on ad-hoc cluster-similarity measures, MD takes principled hypothesis testing in statistics, called *test of homogeneity* [14], to evaluate if multiple clusters of data are generated from homogeneous distributions.

The rest of the thesis is organized as follows: Chapter 2 presents the MGS framework and Chapter 3 the DCM framework. Chapter 4 discusses the "emsemblization" of the DCM framework. Chapter 5 presents the structure-driven crawler for query interfaces and Chapter 6 the model-differentiation based clustering algorithm. Chapter 7 reviews related work. Chapter 8 concludes the thesis.

Chapter 2

Global Evaluation: Matching as Hidden Model Discovery

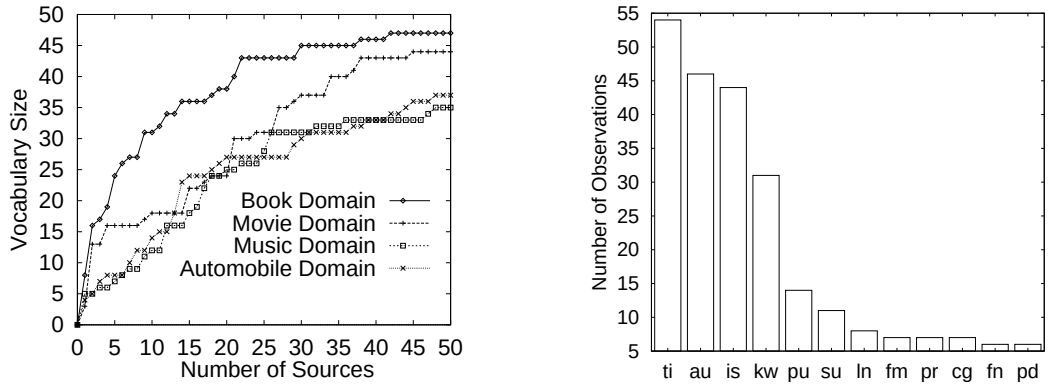
As a first step toward holistic schema matching, we develop the MGS framework with specific focus on simple 1:1 matchings. In particular, we hypothesize the existence of the hidden generative behavior of a schema model, which captures synonym relationships of attributes. This hidden-model hypothesis provides a principled statistical method, hypothesis testing [9], to globally evaluate the confidence of a model (as a statistical hypothesis), given a set of schemas as observations. We thus abstract the schema matching problem as hidden model discovery and develop the MGS framework [37] to realize such a global evaluation strategy.

2.1 Motivation

The "wild" frontier of the deep Web is characterized by its unprecedented scale. As a *challenge*: we often need to match large numbers of sources. As an *opportunity*: ample sources are usually available

domain	*sources*	*all attributes*	*non-rare*
Books	55	47	12
Movies	52	44	12
Music Records	49	35	11
Automobiles	55	37	11

Figure 2.1: Statistics of sources studied.



(a) Vocabulary growth over proliferating sources. **(b)** Frequencies over ranks of attributes.

Figure 2.2: Analyzing schema vocabularies of deep Web sources.

to form a useful "context" of matching. Intuitively, by holistically unifying many sources in the same domain, our statistical approach intends to leverage the opportunity while addressing the challenge.

2.1.1 Deep Web Observations

To understand their characteristics, we performed an informal study of sources on the deep Web. From Web directories, we drew sources in each of the four domains: Books, Music Records, Movies, and Automobiles. In particular, we collected all of invisibleweb.com's sources (in these 4 domains) and most of yahoo.com's without any bias, until reaching about 50 sources in each domain, as Figure 2.1 summarizes[1].

On the one hand, we observe *proliferating sources*: As discussed in Chapter 1, while many Web directories such as invisibleweb.com already list impressive numbers of online sources by manual com-

[1]This dataset is available as the BAMM dataset of the UIUC Web integration repository [17].

pilation, there are certainly many more sources out there. As the Web continues to expand, it will house virtually unlimited numbers of sources in interesting domains.

On the other hand, we also observe *converging vocabularies*: The aggregate schema vocabulary of sources in the same domain tends to converge at a relatively small size. Figure 2.1 summarizes (in the middle column) the sizes of the entire vocabularies of all attributes used in any sources, which are about 40 for each domain. Figure 2.2(a) further analyzes the growth of vocabularies as sources increase in number. The curves clearly indicate the convergence of vocabularies. For instance, for the Books domain, 92% (43/47) attributes are observed at 25^{th} sources, and 98% (46/47) at 35^{th}. Since the vocabulary growth rates (*i.e.*, the slopes of these curves) decrease rapidly, as sources proliferate, their vocabularies will tend to stabilize. (Note that the sources are sorted in the same order as they were collected without any bias.)

In fact, the vocabularies will converge more rapidly if we exclude "rare" attributes. To quantify, let the frequency of an attribute be the number of sources in which it occurs. Figure 2.2(b) orders these frequencies for the book domain over their ranks, with attributes detailed in Figure 2.7. It is interesting but perhaps not surprising to observe that the distribution obeys Zipf's law: The frequencies are inversely proportional to their ranks. Many low-ranked attributes thus rarely occur; Figure 2.2(b) shows only the top 12 attributes (which account for 78% or 230/294 of all the attribute occurrences); most others occur only once. In practice, these rare attributes are likely unimportant in matching since their rareness indicates that very few other sources will find them useful. With such rare attributes (say, below 10% frequencies) excluded, the "useful" vocabularies are much smaller: about 11 attributes per domain (Figure 2.1).

Note that, while vocabularies tend to converge, schema heterogeneity still persists. That is, although Web query interfaces tend to share attributes, they are not universally shared– thus creating the real chal-

lenge of schema matching. In particular, among the top "popular" attributes for books in Figure 2.2(b)–how many different attributes are "synonyms" for the same concepts? We found 5 ({author, last name, first name}, {subject, category}) out of 12, or a significant 42%. We observed similar levels of heterogeneity in other domains as well (see Figure 2.7).

2.1.2 Toward Hidden Model Discovery

These observations lead us to hypothesize the existence of a hidden schema model that probabilistically generates, from a finite vocabulary, the schemas we observed. Intuitively, such a model gives the "structure" of the vocabulary to constrain how instances can be generated. We believe this hypothesis reasonable, since it naturally explains our observations in Section 2.1.1.

The hypothesis sheds new light on a different way for coping with schema matching: If a hidden model does exist, its *discovery* would reveal the vocabulary structure, which will in principle answer "any" schema matching questions. (As an analogy, an English dictionary can semantically relate all English words, subsuming the need for their pairwise correspondence.) Such model-level unification of all attributes in the same domain will subsume their pairwise correspondence (as used in traditional schema matching). We thus propose an approach to holistically matching schemas as hidden model discovery.

2.2 The MGS Framework

As just motivated, we view schema matching as a quest for an underlying model generating the input schemas. That is, our probabilistic approach seeks to treat the schemas as being generated by a random process following a specific distribution. Our goal is thus, given the input schemas as "observations," to reconstruct the hidden generative distribution.

To realize such hidden model discovery, we propose a general framework, MGS, consisting of hypothesis modeling, generation, and selection. We believe the MGS framework is important in its own right: In principle, by application-specific hypothesis modeling, MGS can be applied to capture different types of semantic relationships. Specifically,

1.*Hypotheses Modeling*: To guide the seeking of a hypothetical model, or a *hypothesis*, we start by defining the general structure of such models. Such modeling should essentially capture specific semantics we want to discover. For instance, if we want to find synonyms, a model should explicitly express the relationship of "synonyms." Such modeling will also specify a generative behavior of how schemas can be generated. Such behavior is mainly *probabilistic* (*e.g.*, attributes will be drawn randomly by their "popularity"), although it can also partially be *deterministic* (*e.g.*, no synonyms can be selected together). Effectively, the model forms a statistical distribution, which generates a particular schema with some *instantiation probability*.

2.*Hypotheses Generation*: We then enumerate concrete hypotheses (in the specified abstract model) that are consistent with the observed schemas (with non-zero probabilities). Note that, even with a parameterized structure, there will be a large space of candidate hypotheses to search, for a vocabulary of reasonable size. This generation step helps to focus the search to only those promising hypotheses that are likely to generate the observed schemas.

3.*Hypotheses Selection*: Finally, we select hypotheses that are consistent with the observed schemas with sufficient statistical significance. There are various statistical devices for such hypothesis testing [9]. For instance, we use χ^2 testing in our MGS_{ac} algorithm.

In summary, we propose MGS as a general framework for the hidden model discovery problem: Given a set of schemas I as observations, hypothesize and select the schema models with sufficient

statistical consistency as the generative distributions of I. We next specialize the abstract framework for synonym discovery.

2.3 Synonym-Attribute Discovery

Finding corresponding attributes is a central problem for schema matching; in this chapter, we pursue this problem as *synonym discovery*, *i.e.*, discovering simple 1:1 matchings. The challenge is to find the synonyms among the input attributes, typically without the semantics understanding of those attributes. That is, across different schemas, some attributes (*e.g.*, author and name, or subject and category) are *synonyms* for the same *concepts* (*e.g.*, for the "author" and "subject" concepts respectively). As Section 2.1 motivated, we focus on matching query interfaces for sources in the same domain on the deep Web. Thus, given such schemas, our goal is to discover all the synonym attributes.

Guided by the general MGS framework, we develop Algorithm MGS_{ac} (Figure 2.5), specifically for discovery of synonym attributes as the target question. MGS_{ac} first defines the hypothetical model structure for capturing synonym attributes (Section 2.3.1), generates the model candidates that have non-zero probabilities (Section 2.3.2), and selects the sufficiently consistent ones (Section 2.3.3). Beyond these essential steps, we develop techniques for coping with several real-world issues that complicate our statistical approach (Section 2.3.4). Finally, we put all the components together to present the complete algorithm (Section 2.3.5).

2.3.1 Hypothesis Modeling

Following MGS, we first define the structure of the underlying model. Specifically, we aim at answering the target question of synonym attributes for Web interfaces. (Incidentally, our model can also capture the target question of concept popularity.) We view a query interface as a "flat" schema, or a set of

attributes; *e.g.*, amazon.com has a schema {title, author, $\cdots$}. This simple view is sufficient for our purpose of synonym discovery. In particular, we do not concern complex matching (*e.g.*, author as last and first name), which itself is another interesting target question. In particular, the DCM framework is developed for coping with such complex matchings (Chapter 3).

To reasonably define a simple model, we make several assumptions of how our schemas are generated. (Imagine a human Web developer generates such Web interfaces to bring databases online.) First, *concept mutual-independence*: A query interface contains several different concepts (*e.g.*, "author" or "subject"). We assume that, in generating a schema (which may not contain all concepts), different concepts are selected independently.

Second, *synonym mutual-exclusion*: When multiple synonyms exist for a concept (*e.g.*, author and name), we assume that, in generating a schema, no two synonyms will both be selected. Such duplicated selections will create redundancy and perhaps confusion; our case studies (of real sources; Section 2.1.1) in fact have found no such schemas. As Section 2.3.2 will discuss, this mutual exclusion enables significant pruning of the hypothetical model space.

Third, *non-overlapping concepts*: We assume that different concepts do not overlap in semantics, *i.e.*, no distinct concepts will share attributes. This assumption holds in most cases, when synonyms in the same concept are fully *equivalent*: *e.g.*, concepts {author, name} and {subject, category} do not overlap. Thus this assumption says that all concepts will form an *equivalence partition* of the vocabulary set. However, as our case studies observed (Section 2.4), sometimes an attribute can be a *non-equivalent* synonym to others, and thus participate in distinct concepts– *e.g.*, concepts {author, last name} and {author, first name}, where author corresponds to last name and first name in different "senses." This assumption excludes such cases: Instead of complicating simple synonym equivalence,

such cases can be more systematically treated, by first grouping attributes [last name, first name] and then finding equivalent synonym {author, [last name, first name]} (see Chapter 3).

2.3.1.1 Model Structure

Based on our assumptions, we define a simple model for capturing synonym attributes. Essentially, a model describes how to generate schemas from a vocabulary of attributes. Figure 2.3 visualizes $\mathcal{M}_B$ (an example for book sources) as a two-level tree, for vocabulary $\mathcal{V}_B$ = {author, title, ISBN, subject, category}. To express synonyms, our model partitions all attributes into concepts, or equivalent classes (by the non-overlapping concepts assumption): *e.g.*, C_1: {author}, $\cdots$, C_4: {subject, category} in $\mathcal{M}_B$. The model will generate schemas by, first, independently selecting each concept C_i with probability α_i (by concept mutual-independence). For any selected concept, the model will then choose exactly one of its member attributes A_j with probability β_j (by synonym mutual-exclusion). The model thus generates a schema with all the chosen attributes.

Definition 1: A *schema model* $\mathcal{M}$ is a 4-tuple $(\mathcal{V}, \mathcal{C}, P_c, P_a)$: The *vocabulary* $\mathcal{V}$ is a set of attributes $\{A_1, \cdots, A_n\}$. The *concept partition* $\mathcal{C}$ is a set of *concepts* $\{C_1, \cdots, C_m\}$ that partition $\mathcal{V}$ (*i.e.*, $\mathcal{V} = \cup_{1 \le i \le m} C_i$ and $C_i \cap C_k = \emptyset$). P_c is the *concept probability function*, which determines the probability α_i for including concept C_i in schema generation. P_a is the *attribute probability function*, which determines the probability β_j for selecting attribute A_j, once its concept is included. For every concept C_i: $\sum_{A_j \in C_i} \beta_j = 1$.

Notationally, we will write a model by parenthesizing attributes in concepts with probability annotations, *e.g.*: (Figure 2.3) $\mathcal{M}_B$ = {(author: β_1): α_1, (title: β_2): α_2, (ISBN: β_3): α_3, (subject: β_4, category: β_5): α_4}. When probabilities are not critical in the context, we will simply write $\mathcal{M}_B$ = {(author), (title), (ISBN), (subject, category)}).

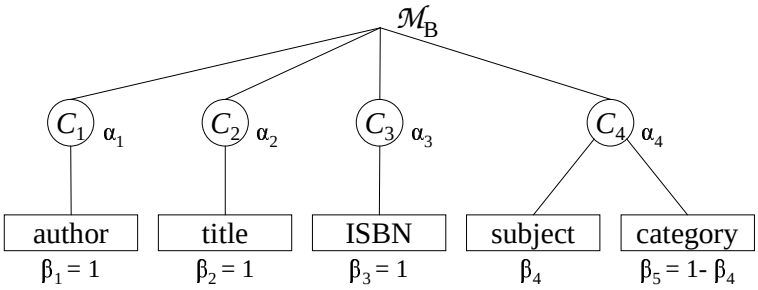


Figure 2.3: An example of schema model $\mathcal{M}_B$.

2.3.1.2 Schema Generation and Observations

We now discuss how a model $\mathcal{M}$ will generate schemas. By Definition 1, $\mathcal{M}$ will simply decide, for each concept C_i, if C_i is included, and if so, select one attribute A_j to represent C_i. This process will generate a schema as a set of attributes.

Example 1: For $\mathcal{M}_B$ in Figure 2.3: Possible schemas (with non-zero probabilities) from $\mathcal{M}_B$ include: I_1 = {author, title, subject, ISBN} and I_2 = {title, category, ISBN}.

Note that a model $\mathcal{M}$, by definition, represents a generative distribution, giving probabilities for any schema that can be generated. We now formalize such probabilities. First, to generate a schema, $\mathcal{M}$ selects concepts to include: By Definition 1, a concept C_i will appear with probability $Pr(C_i|\mathcal{M}) = \alpha_i$ or otherwise $Pr(\neg C_i|\mathcal{M}) = 1 - \alpha_i$.

Next, we consider the probability of picking some attribute: By Definition 1, the probability of selecting an individual attribute A_j in schema generation from $\mathcal{M}$ is:

$$Pr(A_j|\mathcal{M}) = \begin{cases} \alpha_i \times \beta_j, & \exists i : A_j \in C_i \\ 0, & otherwise \end{cases}$$

How about selecting a set of attributes A_1, A_2,.., A_m from $\mathcal{M}$ in any schema? Definition 1 implies this probability as below, where the first condition represents synonym mutual-exclusion and the other

concept mutual-independence.

$$Pr(A_1, A_2, .., A_m|\mathcal{M}) = \begin{cases} 0, \exists j \neq k, \exists i : A_j \in C_i \wedge A_k \in C_i \\ \prod Pr(A_j|\mathcal{M}), otherwise \end{cases}$$

Putting this together, we can derive the probability that $\mathcal{M}$ will generate some schema I, denoted by $Pr(I|\mathcal{M})$. Definition 2 below formalizes this *instantiation probability*. Specifically, $Pr(I|\mathcal{M})$ is the probability of used attributes times the probability of unselected concepts.

Definition 2: For model $\mathcal{M} = (\mathcal{V}, \mathcal{C}, P_c, P_a)$, the *instantiation probability* of a schema $I = \{A_1,...,A_m\}$ is $Pr(I|\mathcal{M}) = Pr(A_1, A_2, .., A_m|\mathcal{M}) \times \prod_{\{C_i|\forall A_j, A_j \notin C_i\}} Pr(\neg C_i|\mathcal{M})$. We say I can be *instantiated* from model $\mathcal{M}$ if $Pr(I|\mathcal{M}) > 0$.

Example 2: Continuing Example 1: we have $Pr(I_1|\mathcal{M}_B) = \alpha_1 \times \alpha_2 \times \alpha_3 \times \alpha_4 \times \beta_4$, $Pr(I_2|\mathcal{M}_B) = (1 - \alpha_1) \times \alpha_2 \times \alpha_3 \times \alpha_4 \times \beta_5$, where $(1 - \alpha_1)$ is the probability that the concept C_1 is not used. However, for I_3 = {author, ISBN, subject, category}, we have $Pr(I_3|\mathcal{M}_B) = 0$, since subject and category both belong to C_4. Thus I_1 and I_2 can be instantiated from $\mathcal{M}_B$, but I_3 cannot.

Our approach seeks to discover the hidden model from many schemas observed (as input). Therefore, we will take a set of schemas I (*e.g.*, the Web sources summarized in Figure 2.1), our input, as *schema observations*. To emphasize that in our input we may observe the same schema several times, we write I as a set of pairs $\langle I_i, B_i \rangle$. Each $\langle I_i, B_i \rangle$ denotes the number of occurrences B_i for each schema I_i.

To discover the hidden model, it is essential to answer: Given model $\mathcal{M}$, how likely will $\mathcal{M}$ generate the schemas in I? (Or, how likely can we observe I, if $\mathcal{M}$ is the hidden model?) It follows Definition 2 that this probability is $Pr(I|\mathcal{M}) = \prod Pr(I_i|\mathcal{M})^{B_i}$. Note that, if $Pr(I|\mathcal{M}) = 0$, it is impossible to observe

I under $\mathcal{M}$. Therefore, we say model $\mathcal{M}$ is *consistent* with observations I, if $Pr(I|\mathcal{M}) > 0$. Thus, the hypothesis generation finds these "consistent models" as candidate hidden models (Section 2.3.2).

Example 3: Continuing Example 2: We may have observations $I = \{\langle I_1, 3\rangle, \langle I_2, 5\rangle\}$, *i.e.*, I_1 3 times and I_2 5 times. Thus, $Pr(I|\mathcal{M}_B) = Pr(I_1|\mathcal{M}_B)^3 \times Pr(I_2|\mathcal{M}_B)^5$. Note $\mathcal{M}_B$ is consistent with I, since $Pr(I_1|\mathcal{M}_B)$ and $Pr(I_2|\mathcal{M}_B)$ are both non-zero (Example 2).

2.3.2 Hypothesis Generation

Guided by the second step of the MGS framework, we now generate candidate models that are likely to be sufficiently consistent (which Section 2.3.3 will determine) with the input observations I. It is clear that any candidate $\mathcal{M}$ has to be at least consistent with I, *i.e.*, $Pr(I|\mathcal{M}) > 0$, so that I is at least possible under $\mathcal{M}$ (Section 2.3.1). This section focuses on constructing such models.

Intuitively, we want to reconstruct $\mathcal{M}$ from our given observations I. Using a statistical approach, we assume the observations are *unbiased* and *sufficient*. First, by the *unbiased* assumption, we will observe (or collect) a schema I with a frequency proportional to how likely I will be generated under $\mathcal{M}$, *i.e.*, $Pr(I|\mathcal{M})$. (*e.g.*, we will not collect *only* schemas that contain author, since that would be biased.) Second, by the *sufficient* assumption, our observations will be large enough so that every possible schema is represented in I. We use these assumptions to estimate the probability parameters (P_a and P_c) of a candidate model. In practice, the sufficient assumption is likely not to be satisfied; we discuss techniques for dealing with "the real world" in Section 2.3.4.

Our goal in hypothesis generation is, given I, to construct models $\mathcal{M} = (\mathcal{V}, \mathcal{C}, P_c, P_a)$ so that $Pr(I|\mathcal{M}) > 0$. To begin with, we determine $\mathcal{V}$: By our above assumptions, $\mathcal{V} = \cup I_i$. Since every possible schema occurs in I, so does every attribute in $\mathcal{V}$. On the other hand, even if the observations are not perfect, for our purpose of matching, we do not care about any "phantom" attributes that have

not occurred in any input source. Thus, our model will capture only attributes that are used by at least one schema (in I).

Next, having determined $\mathcal{V}$, we complete the model ($\mathcal{V}$, $\mathcal{C}$, P_c, P_a) by constructing first the concept partition $\mathcal{C}$ (Section 2.3.2.1), and then the probabilities P_c, P_a (Section 2.3.2.2).

2.3.2.1 Building Concept Partitions

Given the vocabulary set $\mathcal{V}$, we first construct a concept partition $\mathcal{C}$ for a candidate model. By Definition 1, $\mathcal{C}$ is a partition of $\mathcal{V}$. It is clear that, given $\mathcal{V}$, there can easily be a large number of possible partitions. The number of partitions for an *n*-set is called a Bell number $B(n)$, which has an exponential generating function and satisfies recursive relation $B(n+1) = \sum_{k=0}^{n} B(k)\binom{n}{k}$. A vocabulary with, say, 12 attributes will thus have 4213597 possible concept partitions (and as many possible models).

To cope with the large space, it is thus critical to focus on only concept partitions that can lead to consistent models ($\mathcal{M}$, such that $Pr(I|\mathcal{M}) > 0$). These consistent models form the *hypothesis space* with respect to I. Our case studies show that the "consistent" condition can prune the search space to a very small number of models. For instance, in the book domain, we only have 20 models left in the hypothesis space with 12 attributes.

Not all concept partitions are useful for constructing a consistent model, since not every model (with arbitrary concept partitions) can generate a observed schema. In particular, as Example 2 showed, I_3 cannot be observed under $\mathcal{M}_B$, or $Pr(I_3|\mathcal{M}_B) = 0$, since subject and category are both synonyms in C_4 (Figure 2.3). Thus, if I_3 is in I as part of our input schema, we will not consider $\mathcal{M}_B$, since it will be inconsistent with I, *i.e.*, $Pr(I|\mathcal{M}_B) = 0$ as $Pr(I_3|\mathcal{M}_B) = 0$.

We can easily generalize this idea to focus on models that will not "contradict" any observed schema I. In such models, no concept will contain two attributes A_j and A_k that are used by the same schema in

I. (In Example 2, $\mathcal{M}_B$ is not good for I_3, since $\mathcal{M}_B$ contains C_4 with attributes subject and category both from I_3.) That is, we will construct consistent models by using only *consistent concepts*, which do not contain any *co-occurring attributes* from any schema in I. Property 1 formalizes this idea.

Property 1: Given observations I with vocabulary $\mathcal{V}$, let $\mathcal{C} = \{C_1, \cdots, C_m\}$ be a concept partition for vocabulary $\mathcal{V}$. Any model $\mathcal{M}$ constructed from $\mathcal{C}$ will be inconsistent with I, or $Pr(I|\mathcal{M}) = 0$, if for some attributes A_j and A_k, both of the following hold:

1. $\exists$ schema $I_i \in I$, such that $A_j \in I_i$ and $A_k \in I_i$.
2. $\exists$ concept $C_i \in \mathcal{C}$, such that $A_j \in C_i$ and $A_k \in C_i$.

Based on Property 1, we use a two-step process to build the hypothesis space (of consistent models) using consistent concepts as building blocks. (For instance, $\mathcal{M}_B$ in Figure 2.3 is built upon concepts C_1, $\cdots$, C_4.) Step 1, CONSISTENTCONCEPTSCONSTRUCTION, will first find all consistent concepts, and Step 2, BUILDHYPOTHESISSPACE, will then build consistent models accordingly. These two procedures are used to build the initial hypothesis space in Algorithm MGS_{ac}.

In Step 1, we can translate the problem of finding consistent concepts into finding all cliques in an attribute "co-occurrence graph" [23]. Specifically, we construct a *concept network* from our observations I: In this graph, a node represents an attribute, and an edge exists between attributes A_j and A_k if and only if they do not co-occur in any schema I in I. Thus, non-cooccurring attributes will be connected with an edge– Precisely such attributes will form consistent concepts. However, a concept can be of any number of attributes. Therefore, we look for cliques for any size in the graph to construct consistent concepts.

Example 4: Consider observations I in Example 3. From I, we can derive its concept network in Figure 2.4. In particular, author and title do not have an edge because they co-occur in I_1. Author and category have an edge since they do not co-occur in any schema.

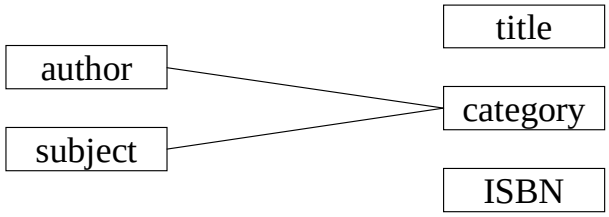


Figure 2.4: An example concept network.

Further, what can be consistent concepts? There are 7 cliques in Figure 2.4: {author}, {title}, {subject}, {category}, {ISBN}, {author, category}, and {subject, category}. Any clique represents a cluster of non-cooccurring attributes, and therefore is a consistent concept (by Property 1). In particular, some of these concepts, such as {author} and {subject, category}, are part of $\mathcal{M}_B$, which is consistent with I (as Example 3 explained).

In Step 2, we use the consistent concepts just obtained as the building blocks for constructing consistent models. Since all the concepts in a model partition its vocabulary set $\mathcal{V}$, this step is essentially a classic set cover problem [23], with the covering subsets being non-overlapping. That is, given some subsets (the consistent concepts) of set $\mathcal{V}$, we want to select some non-overlapping subsets to cover $\mathcal{V}$. Below we illustrate the result of constructing all the consistent models as the hypothesis space, which concludes our hypothesis generation step in this section.

Example 5: Given the consistent concepts in Example 4, we can construct a consistent model $\mathcal{M}_1$ = {(author), (title), (ISBN), (subject), (category)}, since the five concepts partition the vocabulary. We can find all the other consistent models: $\mathcal{M}_2$ = {(author), (title), (ISBN), (subject, category)} and $\mathcal{M}_3$ = {(author, category), (title), (ISBN), (subject)}. The hypothesis space is therefore $\{\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3\}$.

2.3.2.2 Building Probability Functions

We have generated all the consistent models, which form the hypothesis space. However, these models are still incomplete: As a 4-tuple $(\mathcal{V}, \mathcal{C}, P_c, P_a)$, $\mathcal{M}$ has yet to determine the probability functions P_c

and P_a, although $\mathcal{V}$ and $\mathcal{C}$ are specified. Recall our ultimate goal is to discover those hidden models that are sufficiently consistent with input I. So far, for each consistent model $\mathcal{M}$, by building upon only consistent concepts, we guarantee that $Pr(I|\mathcal{M})$ is not *necessarily* zero. Therefore, there exist P_c and P_a assignments for $\mathcal{M}$, such that $Pr(I|\mathcal{M}) > 0$.

To complete each of these consistent models, we still need to specify P_c and P_a– clearly these probabilities should further maximize $Pr(I|\mathcal{M})$. The reason is that with the assumptions of unbiased and sufficient input data, the values of P_c and P_a must be the ones that make the model the most consistent with the data. The "consistency" is reflected as the instantiation probability. So the most consistent model is corresponding to the model with the highest probability. Thus, we have an optimization problem to find

$$\max_{P_c,P_a} Pr(I|\mathcal{M}(\mathcal{V},\mathcal{C},P_c,P_a)), \tag{2.1}$$

which is essentially the *maximum likelihood estimation* problem, for given $\mathcal{V}$ and $\mathcal{C}$.

Example 6: Continue Example 5, where we showed $\mathcal{M}_2$ as one of the consistent models. To completely specify $\mathcal{M}_2$, we need to determine P_c and P_a to maximize $Pr(I|\mathcal{M}_2)$ (for I given in Example 3).

As Example 3 derives (note $\mathcal{M}_2$ and $\mathcal{M}_B$ are the same model): $Pr(I|\mathcal{M}_2) = Pr(I_1|\mathcal{M}_2)^3 \times Pr(I_2|\mathcal{M}_2)^5 = \alpha_1^3 \times (1-\alpha_1)^5 \times \alpha_2^8 \times \alpha_3^8 \times \alpha_4^8 \times \beta_4^3 \times \beta_5^5$. We apply maximum likelihood estimation to select those α's and β's that can maximize $Pr(I|\mathcal{M}_2)$. The result is $\alpha_1 = 0.375, \alpha_2 = 1, \alpha_3 = 1, \alpha_4 = 1, \beta_4 = 0.375$, and $\beta_5 = 0.625$.

In maximum likelihood estimation of functions P_c and P_a, we are effectively estimating parameters α_i and β_j (Definition 1). Since concepts are independently selected (the concept mutual independence assumption of Section 2.3.1), each α_i can be estimated independently. We can also derive the solution

for β_j based on [9], since β_j in a concept C_i form a multinomial distribution. Therefore, for any schema model, Equation 2.1 has the closed-form solutions:

$$\alpha_i^* = \frac{\sum_{A_j \in C_i} O_j}{|I|}, \beta_j^* = \frac{O_j}{\sum_{A_j \in C_i} O_j}$$

where O_j is the *frequency* of attribute A_j in observations I (*i.e.*, the number of schemas that contain A_j), and $|I|$ is the total number of schemas in I.

2.3.3 Hypothesis Selection

Guided by the third step of the MGS framework, we need to select sufficiently consistent hypotheses. After hypothesis generation, a hypothesis is a determined model (distribution) $\mathcal{M} = (\mathcal{V}, \mathcal{C}, P_c, P_a)$. We propose to apply χ^2 hypothesis testing to quantify how consistent the schema model is with the data. Below we briefly introduce χ^2 testing [9].

Suppose we have n independent observations (schemas) and in each observation, precisely one of r events (schemas with non-zero probability), $I_1, .., I_r$ must happen, and their respective probabilities are $p_1, .., p_r$, with $\sum_{j=1}^{r} p_j = 1$. Suppose that $p_{10}, .., p_{r0}$ are the respective instantiation probabilities of the observed $I_1, .., I_r$ with respect to the tested model $\mathcal{M}$, with $\sum_{j=1}^{r} p_{j0} = 1$. We want to test the hypothesis $p_1 = p_{10}, .., p_r = p_{r0}$ by considering the statistic

$$D^2 = \sum_{j=1}^{r} \frac{(B_j - np_{j0})^2}{np_{j0}}$$

where n is essentially $|I|$. It can be shown that D^2 has asymptotically a χ^2 distribution with $r-1$ degrees of freedom. Again a test of the null hypothesis $H : p_1 = p_{10}, .., p_r = p_{r0}$ at the $100a\%$ significance level is obtained by choosing a number b such that $Pr\{D^2 > b\} = a$, where D^2 has the χ^2 distribution with $r-1$ degrees, and rejecting the hypothesis if a value of D^2 greater than b is actually observed.

Example 7: Assume we have observations $I = \{\langle I_1, 6\rangle, \langle I_2, 3\rangle, \langle I_3, 1\rangle\}$, with $I_1 = \{$author, subject$\}$, $I_2 = \{$author, category$\}$, and $I_3 = \{$subject$\}$. Our goal is to select the schema model at the significance level 0.05. The hypothesis generation step will output two hypotheses (models):

$\mathcal{M}_1 = \{$(author:1):0.6, (subject:0.7, category:0.3):1$\}$ and

$\mathcal{M}_2 = \{$(author:1):0.6, (subject:1):0.7, (category:1):0.3$\}$.

We first consider $\mathcal{M}_1$. Four schemas can be instantiated from $\mathcal{M}_1$: $\{$subject$\}$, $\{$category$\}$, $\{$author, subject$\}$, and $\{$author, category$\}$ with instantiation probabilities 0.28, 0.12, 0.42, and 0.18 respectively. Thus, the computation of D^2 is: $D^2(\mathcal{M}_1) = \frac{(1-10*0.28)^2}{10*0.28} + \frac{(0-10*0.12)^2}{10*0.12} + \frac{(6-10*0.42)^2}{10*0.42} + \frac{(3-10*0.18)^2}{10*0.18} \doteq 3.93$ with freedom degree 3. The χ^2 distribution table shows $Pr(D^2 > 7.815) = 0.05$ at that freedom degree. Since $3.93 < 7.815$, we accept this hypothesis and consider it as a sufficiently consistent schema model.

$\mathcal{M}_2$ is processed in the same way. Eight schemas can be instantiated from $\mathcal{M}_2$: $\{\}$, $\{$author$\}$, $\{$subject$\}$, $\{$category$\}$, $\{$author, subject$\}$, $\{$author, category$\}$, $\{$subject, category$\}$, and $\{$author, subject, category$\}$ with probabilities 0.084, 0.126, 0.196, 0.036, 0.294, 0.054, 0.084, and 0.126 respectively. Then we have $D^2(\mathcal{M}_2) \doteq 20.20$ with freedom degree 7. The χ^2 distribution table shows $Pr(D^2 > 14.067) = 0.05$. Since $20.20 > 14.067$, we should not select $\mathcal{M}_2$. Therefore, hypothesis selection will select $\mathcal{M}_1$ as the schema model.

2.3.4 Dealing With the Real World

We presented the overall process of Algorithm MGS$_{ac}$, guided by the general principles of the MGS framework. Further, there are often "real-world" issues on data observations that can compromise a statistical approach like ours. We find that, specifically for schema matching, the key challenge is the extremely "unbalanced" attribute distribution. We observed a Zipf-like distribution (Figure 2.2b) of attributes in our analysis of deep Web sources (Section 2.1).

Challenges arise on the either end of this Zipf distribution: On one hand, the *head-ranked* attributes (*e.g.*, ti and au in Figure 2.2b) are extremely frequent, occurring in almost every schema: Their occurrences tend to dominate any models and thus render these models indiscriminate under hypothesis testing (as Section 2.3.3 developed). Section 2.3.4.3 addresses dominating attributes with incremental consensus projection to isolate their effects.

On the other hand, the *tail-ranked* attributes (*e.g.*, those not shown in Figure 2.2b) are extremely rare, often occurring only once in some schema. Their occurrences in observations (while rare) tend to "confuse" our statistical approach that asserts sufficient samples. In principle, a rare attribute A can appear in many concepts (by combining with other attributes in schema generation). Also, as A being rare, these "A-schemas" are unlikely to be observed in I if it is not arbitrarily large– Thus A will compromise a statistical approach for the lack of schemas. Section 2.3.4.1 addresses rare attributes with attribute selection.

Together, this *head-often, tail-rare* attribute distribution will imply similar non-uniformness of schemas. Thus, some schemas (with rare attributes) will be extremely rare too. Our hypothesis testing essentially relies on estimating schema frequencies B_j (Section 2.3.3). A rare schema I occurring only once in I tends to result in an *overestimated* frequency, or I needs to be arbitrarily large to justify I's only occurrence being sufficiently rare. Section 2.3.4.2 addresses rare schemas by "smoothing."

2.3.4.1 Attribute Selection

Rare attributes can confuse a statistical approach, with their lack of complete schemas in our observations I. Such rare attributes will require virtually arbitrarily large I to give them sufficient context. That is, for these rare attributes, I is unlikely to be sufficient to statistically "explain" their properties– Thus, our sufficient assumption (Section 2.3.2) is unlikely to hold for such attributes. To draw valid

statistical results, our approach is to systematically remove rare attributes– they are effectively "noise" in our setting.

Fortunately, these rare attributes may indeed be unimportant in schema matching. As Section 2.1.1 explained, with Zipf distribution, most rare attributes occur in only one source. Thus, few other sources will find these attributes useful in query mediation or data exchange. (A mediator will not be likely to support such attributes; they are neither "mediatable" nor "exchangeable.") We believe it is naturally justified to remove rare noise in matching.

We believe systematic *attribute selection* will be crucial for finding attribute subsets, for which robust statistical results can be achieved. We use a frequency-based pruning to select only frequent attributes into vocabulary $\mathcal{V}$ (Section 2.3.2), as a procedure ATTRIBUTESELECTION in Algorithm MGS_{ac} (Figure 2.5). Specifically, we select an attribute A_j if its observation frequency $O_j \geq f$, where f is a given threshold set as 10% in our experiments. While this empirical value works well (Section 2.4), further investigation is clearly necessary to automate threshold selection.

2.3.4.2 Rare Schema Smoothing

Our observations I may contain infrequent schemas I that are presumably rare, as explained earlier. In particular, the χ^2 testing (Section 2.3.3) evaluates the difference between the estimated probabilities $Pr(I|\mathcal{M})$ and the observed frequencies B_j. For infrequent schemas, such a difference will significantly distort the closeness of D^2 to the χ^2 distribution, which may influence the result of hypothesis selection.

Example 8: Suppose our observations $I = \{\langle I_1, 45\rangle, \langle I_2, 5\rangle, \langle I_3, 2\rangle, \langle I_4, 1\rangle\}$, with $I_1 = \{\text{author}\}$, $I_2 = \{\text{last name}\}$, $I_3 = \{\text{author, price}\}$, and $I_4 = \{\text{price}\}$. The hypothesis generation will find three hypotheses:

$\mathcal{M}_1$={(author:.9, last name:.1):.98, (price:1):.06}

$\mathcal{M}_2$={(author:1):.89, (last name:.62, price:.38):.15}

$\mathcal{M}_3$={(author:1):.89, (last name:1):.09, (price:1):.06}.

The probabilities of I_4 in $\mathcal{M}_1$, $\mathcal{M}_2$, and $\mathcal{M}_3$ are .0012, .0064, and .0058 respectively, which indicates I_4 a rare schema. The χ^2 testing will in fact reject all three models (at the significance level 0.05).

Note that even the correct model $\mathcal{M}_1$ does not pass the test, simply because the early observation of the rare schema I_4 results in an unreliable estimation of its probability. Thus the rare schema disturbs the result.

We cope with this problem by *rare schema smoothing*: Instead of regarding each possible schema I_j as an individual event (Section 2.3.3), we will aggregate infrequent schemas into a conceptual event I_{rare}, whose probability is the sum of the probabilities of its members. Such aggregation will smooth the overestimation in frequency counting, thus giving a more reliable probability indication [3]. We will then take χ^2 testing on those frequent events plus I_{rare}.

The key issue is then how to determine whether a schema I_j is rare. Our basis is its frequency in observations I (with size $|I|$), since the real probability is hidden to be discovered. We apply two criteria: 1) If not observed in I, I_j is rare. 2) If observed, I_j is rare if $Pr(I_j|\mathcal{M}) \times |I| < T_{smooth}$, where T_{smooth} is a threshold (dynamically determined).

We further develop adaptive thresholding of T_{smooth} in smoothing, as a procedure DYNAMICSELECTION in Algorithm MGS_{ac} (Figure 2.5): During hypothesis selection (Section 2.3.3), we test the hypotheses with increasing thresholds until reaching at least one qualified hypothesis. (Implicitly, we are applying our motivating assertion that there must exist a correct hidden model.) Otherwise, it will output all the hypotheses, since they are not distinguishable (and at least one must be correct). Empirically, we start the adaptive thresholding at $T_{smooth} = 0.2$ with a step size 0.1, and stop at 1.0, which works well (Section 2.4).

2.3.4.3 Consensus Projection

Straightforward testing cannot always distinguish models that share a dominating "consensus" (which makes other differences insignificant). As explained earlier, the head-ranked attributes often dominate the testing and thus all these models may agree on the "structure" of these attributes– Such *consensus* can be recognized (for early conclusion) and projected (for isolating dominating attributes). Note that we assume a consensus must be correct, based on our motivating assertion that there exists at least a correct model.

Example 9: Suppose our observations are $I = \{\langle I_1, 45\rangle, \langle I_2, 6\rangle, \langle I_3, 2\rangle, \langle I_4, 4\rangle\}$ with $I_1 = \{\text{title}\}$, $I_2 = \{\text{title, subject}\}$, $I_3 = \{\text{title, subject, price}\}$, and $I_4 = \{\text{title, category}\}$. Hypothesis generation will output three hypotheses:

$\mathcal{M}_1$={(title:1):1, (subject:.67, category:.33):.21, (price:1):.035}

$\mathcal{M}_2$={(title:1):1, (subject:1):.14, (category:.67, price:.33):.11}

$\mathcal{M}_3$={(title:1):1, (subject:1):.14, (category:1):.07, (price:1):.035}.

The χ^2 hypothesis testing will reject all three models at the significance level 0.05. In fact, their D^2 values are not distinguishable, due to the highly frequent attribute title, which dominates the χ^2 testing. However, it is clear that they all share a "consensus" on title.

We thus propose *consensus projection* for recognizing and extracting consensuses (or shared concepts across models), so that hypothesis testing will better focus on models' distinctions. Note that the soundness of such a projection (of consensus concepts) follows our concept mutual independence assumption (Section 2.3.1).

Specifically, consensus projection will extract the consensus from all the models in the hypothesis space. Also it will extract the consensus attributes from the observed schemas and aggregate the projected schemas that become identical. The projection and aggregation will result in a new set of input

schemas, which are used for the re-estimation of the parameters of the projected models. Such a projection can be repeated, since more consensuses will gradually emerge as the algorithm progresses. We can then discover the final models incrementally by projecting consensuses in progressive iterations. We thus structure Algorithm MGS_{ac} as an iterative framework, as Section 2.3.5 will discuss.

Example 10: Continuing Example 9: We recognize concept (title) as the consensus. We thus perform consensus projection to extract (title) from all hypotheses and attribute title from all schemas in I.

So we have $\mathcal{H}^* = \pi_{\{\text{subject,category,price}\}}(\mathcal{H})$, with

$\mathcal{M}_1^*$={(subject:.67, category:.33):1, (price:1):.17}

$\mathcal{M}_2^*$={(subject:1):.67, (category:.67, price:.33):.5}

$\mathcal{M}_3^*$={(subject:1):.67, (category:1):.33, (price:1):.17}

and $I^* = \pi_{\{\text{subject,category,price}\}}(I) = \{\langle I_2^*, 6\rangle, \langle I_3^*, 2\rangle, \langle I_4^*, 4\rangle\}$ with $I_2^* = \{\text{subject}\}$, $I_3^* = \{\text{subject, price}\}$, and $I_4^* = \{\text{category}\}$. I_1^* is empty after projection and thus removed. The new parameters of $\mathcal{M}^*$ are estimated from I^* with maximum likelihood estimation. The χ^2 testing will select $\mathcal{M}_1^*$ (and reject others) at the significance level 0.05.

2.3.5 Putting It All Together: Algorithm MGS_{ac}

For solving the target question of synonym attributes, Algorithm MGS_{ac} (Figure 2.5) consists of two phases: building the initial hypothesis space and iteratively discovering the hidden models. The first phase selects the attributes as the vocabulary (Section 2.3.4.1) and builds the hypothesis space (Section 2.3.2.1). The iterative process is based on consensus projection (Section 2.3.4.3): In each iteration, it projects the consensus, re-estimates the parameters (Section 2.3.2.2), and tests the hypotheses (Section 2.3.3) with the smoothing technique (Section 2.3.4.2).

Example 11: Consider the Books domain sources listed in Figure 2.7. The iterative process is illustrated in Figure 2.6. In the first iteration, the consensus consists of concepts (ti), (is), (kw), (pr), (fm), and (pd). The DYNAMICSELECTION function will select four hypotheses as *selectedH* with T_{smooth} as 0.5, which are listed in the third column of the 1st iteration of Figure 2.6. In the second iteration, the consensus consists of concept (pu). The DYNAMICSELECTION function will select two hypotheses among the four in the 1st iteration. In the third iteration, the consensus is (su, cg) and DYNAMICSELECTION cannot find any passing hypothesis with all the T_{smooth}'s. Therefore, the algorithm will stop and output two discovered schema models: $\mathcal{M}_1$ = {(ti), (is), (kw), (pr), (fm), (pd), (pu), (su, cg), (au, ln), (fn)} and $\mathcal{M}_2$ = {(ti), (is), (kw), (pr), (fm), (pd), (pu), (su, cg), (au, fn), (ln)}, where the parameters α's and β's are omitted.

The time complexity of MGS_{ac} is exponential with respect to the number of attributes. For instance, the complexity of CONSISTENTCONCEPTSCONSTRUCTION is exponential since the clique problem is NP-complete. Similarly, the steps of BUILDHYPOTHESISSPACE and DYNAMICSELECTION are both exponential. Since schema matching is typically done "off-line," such computation time may still be tolerable in most situations. For instance, in our experimental setting (Section 2.4), the running time is typically within one minute. Further, our observation in Section 2.1 indicates that in practice the computation is likely to scale to many sources: Even with more sources, their aggregate vocabulary tends to converge– The growth of attributes and thus the corresponding computation cost are likely to stop at some point. Nevertheless, it is certainly a real issue to explore more efficient algorithms, as Section 8 discusses.

Algorithm: MGS_{ac}:
Input: SchemaSet I, SignificanceLevel a
Output: Schema Model Hypotheses $\mathcal{H}$
begin:
1 /* initial hypothesis generation */
2 $\mathcal{V}$ = ATTRIBUTESELECTION($\bigcup I_i$)
3 C = CONSISTENTCONCEPTSCONSTRUCTION(I)
4 $\mathcal{H}$ = BUILDHYPOTHESISSPACE(C)
5 /* iterative framework */
6 **while** true
7 *conAttrs* = attributes in the consensus of $\mathcal{H}$
8 **if** $conAttrs = \emptyset$ or $\mathcal{V} = conAttrs$
9 **return** the initial models of $\mathcal{H}$
10 **else**
11 /* consensus projection */
12 $\mathcal{V} = \mathcal{V} - conAttrs$; $I^* = \pi_{\mathcal{V}}(I)$; $\mathcal{H}^* = \pi_{\mathcal{V}}(\mathcal{H})$
13 /* maximum likelihood estimation */
14 **for** each $\mathcal{M}$ in $\mathcal{H}^*$
15 estimate parameters α, β of $\mathcal{M}$ using I^*
16 /* hypothesis selection */
17 *selectedH* = DYNAMICSELECTION($\mathcal{H}^*$)
18 /* new hypothesis space for next iteration */
19 $\mathcal{H} = selectedH$
end

Figure 2.5: Algorithm MGS_{ac}.

2.4 Case Studies

To evaluate the MGS_{ac} framework, we test it with four domains of sources on the deep Web. We design two suites of metrics to quantify the accuracy of both the model itself and its ability to answer the target questions. The experimental results show remarkable accuracy for both metrics.

2.4.1 Experiment Setup

We collected over 200 sources over four domains as stated in Section 2.1.1. For each source, we manually extracted attributes from its query interface and did some straightforward preprocessing to merge attributes of slight textual variations (*e.g.*, author's name and author). (This dataset is available as the BAMM dataset in the UIUC Web Integration Repository [17].) Thus, we focus on discovering synonym

kth	*consensus*	*hypotheses pass kth iteration*	T_{smooth}
1st	(ti),(is),(kw), (pr),(fm),(pd)	{(au:0.85,ln:0.15):0.98,(pu:1):0.25,(su:1):0.2,(cg:1):0.13,(fn:1):0.11}	0.5
		{(au:0.85,ln:0.15):0.98,(pu:1):0.25,(su:0.61,cg:0.39):0.33,(fn:1):0.11}	
		{(au:0.88,fn:0.12):0.95,(pu:1):0.25,(su:1):0.2,(cg:1):0.13,(ln:1):0.15}	
		{(au:0.88,fn:0.12):0.95,(pu:1):0.25,(su:0.61,cg:0.39):0.33,(ln:1):0.15}	
2nd	(pu)	{(au:0.85,ln:0.15):0.98,(su:0.61,cg:0.39):0.33,(fn:1):0.11}	0.6
		{(au:0.88,fn:0.12):0.95,(su:0.61,cg:0.39):0.33,(ln:1):0.15}	
3rd	(su,cg)	{(au:0.85,ln:0.15):1,(fn:1):0.11}	1.0
		{(au:0.88,fn:0.12):0.96,(ln:1):0.15}	
4th	$\emptyset$		

Figure 2.6: Process of discovering schema model for the Books domain.

attributes and consider such attribute extraction and preprocessing as independent tasks. In particular, in our later development, we developed automatic techniques for extracting attribute information from query interfaces [72] and preprocessing techniques to merge syntactically similar attributes in the DCM framework (Chapter 3).

In the experiments, we select the attributes using the approach proposed in Section 2.3.4.1 with threshold $f = 10\%$. The attributes passing that threshold are listed in Figure 2.7. Also, in the experiments we assume 0.05 as the significance level of χ^2 hypothesis testing. In practice, the threshold and significance level can be specified by users.

2.4.2 Metrics

We propose two suites of metrics for different purposes. The first suite is generic since it measures how the hypothesized schema model is close to the correct schema model written by human experts. The second suite of metrics is specific in the sense that it measures how well the hypothesized schema model can answer the target questions.

First, we introduce the notion of *correct schema model*. A correct schema model $\mathcal{M}_c$ is a schema model where attributes are correctly partitioned into concepts. Since it is difficult and unreliable (even

for human experts) to specify the ideal probability parameters, we assign them using maximum likelihood estimation, which is consistent with the "unbias" and "sufficient" assumptions in Section 2.3.2.

The purpose of the first suite of metrics is to compare two models (or distributions). We view each distribution as a set of schemas (instantiated from that distribution), associated with a probability (or member frequency). Thus, we adopt precision and recall to measure this "member frequency". We define $Ins(\mathcal{M})$ as the set of all schemas that can be instantiated from $\mathcal{M}$. Precision is designed to measure the portion of the hypothesized set $Ins(\mathcal{M}_h)$ that is correct. In our case, the correct part is the intersection of $Ins(\mathcal{M}_h)$ and $Ins(\mathcal{M}_c)$, denoted by S. So the *model precision* is:

$P_M(\mathcal{M}_h, \mathcal{M}_c) = \frac{\sum_{I \in S} Pr(I|\mathcal{M}_h)}{\sum_{I \in Ins(\mathcal{M}_h)} Pr(I|\mathcal{M}_h)} = \sum_{I \in S} Pr(I|\mathcal{M}_h)$, where $\sum_{I \in Ins(\mathcal{M})} Pr(I|\mathcal{M}) = 1$ for any model $\mathcal{M}$. Similarly, *model recall* measures the portion of $\mathcal{M}_c$ that is contained in $\mathcal{M}_h$, which is $R_M(\mathcal{M}_h, \mathcal{M}_c) = \sum_{I \in S} Pr(I|\mathcal{M}_c)$.

Example 12: Consider Example 7, we can see that the correct schema model is actually $\mathcal{M}_1$ and thus both model precision and recall of $\mathcal{M}_1$ are 1.0. Now consider $\mathcal{M}_2$, although it is rejected, we still can measure it as an example. Example 7 has shown the schemas and instantiation probabilities of $Ins(\mathcal{M}_2)$ and $Ins(\mathcal{M}_c)$. So S contains four schemas: {subject}, {category}, {author, subject} and {author, category}. Then we can compute the model precision and recall as $P_M(\mathcal{M}_2, \mathcal{M}_c) = 0.196 + 0.036 + 0.294 + 0.054 = 0.58$ and $R_M(\mathcal{M}_2, \mathcal{M}_c) = 0.28 + 0.12 + 0.42 + 0.18 = 1$.

The second suite of metrics measures how correct the model in answering the target questions. In our case, the target question is to ask for the synonyms of attributes. Specifically, we imagine there is a "random querier" who will ask for the synonyms of each attribute according to the probability of that attribute. The model will answer each question by returning the set of synonyms of the queried attribute in that model. We define $Syn(A_j|\mathcal{M})$ as the set of synonyms of attribute A_j in model $\mathcal{M}$. To

compare two synonym sets, precision and recall are again applied. Given the correct model $\mathcal{M}_c$ and a hypothesized model $\mathcal{M}_h$, the precision and recall of the synonym sets of attribute A_j are:

$P_{A_j}(\mathcal{M}_h, \mathcal{M}_c) = \frac{|Syn(A_j|\mathcal{M}_c) \cap Syn(A_j|\mathcal{M}_h)|}{|Syn(A_j|\mathcal{M}_h)|}$ and

$R_{A_j}(\mathcal{M}_h, \mathcal{M}_c) = \frac{|Syn(A_j|\mathcal{M}_c) \cap Syn(A_j|\mathcal{M}_h)|}{|Syn(A_j|\mathcal{M}_c)|}$.

For this "random querier," more frequently observed attributes have higher probabilities to be asked. Thus we compute the weighted average of all the P_{A_j}'s and R_{A_j}'s as the *target precision* and *target recall*. The weight is assigned as a normalized probability of the attributes. That is, for attribute A_j, the weight $w_j = \frac{Pr(A_j|\mathcal{M})}{\sum_{A_j} Pr(A_j|\mathcal{M})} = \frac{\alpha_i \times \beta_j}{\sum_{A_j} \alpha_i \times \beta_j} = \frac{O_j}{\sum O_k}$ ($\alpha_i \times \beta_j = \frac{O_j}{|I|}$ according to the formulae in Section 2.3.2.2). Therefore, *target precision* and *target recall* of $\mathcal{M}_h$ with respect to $\mathcal{M}_c$ are defined as:

$$P_T(\mathcal{M}_h, \mathcal{M}_c) = \sum_{A_j \in \mathcal{V}_h} \frac{O_j}{\sum O_k} P_{A_j}(\mathcal{M}_h, \mathcal{M}_c)$$

$$R_T(\mathcal{M}_h, \mathcal{M}_c) = \sum_{A_j \in \mathcal{V}_c} \frac{O_j}{\sum O_k} R_{A_j}(\mathcal{M}_h, \mathcal{M}_c),$$

where $\mathcal{V}_h$ and $\mathcal{V}_c$ are the vocabulary sets of $\mathcal{M}_h$ and $\mathcal{M}_c$.

Example 13: In Example 7, the target precision and recall of $\mathcal{M}_1$ are both 1.0 since $\mathcal{M}_1$ is the correct schema model. For $\mathcal{M}_2$, we have $P_{\text{author}}(\mathcal{M}_2, \mathcal{M}_c) = 1$ and $R_{\text{author}}(\mathcal{M}_2, \mathcal{M}_c) = 1$ since author is correctly partitioned in $\mathcal{M}_2$. However, for subject, we have $Syn(\text{subject}|\mathcal{M}_c) = \{\text{category}\}$ and $Syn(\text{subject}|\mathcal{M}_2) = \emptyset$. Therefore $P_{\text{subject}}(\mathcal{M}_2, \mathcal{M}_1) = 1$ and $R_{\text{subject}}(\mathcal{M}_2, \mathcal{M}_1) = 0$. We do the same measurement on category and then compute the weighted average. The occurrences of author, subject, and category are 9, 7, and 3 respectively. Thus, the results are $P_T(\mathcal{M}_2, \mathcal{M}_c) = \frac{9}{19} \times 1 + \frac{7}{19} \times 1 + \frac{3}{19} \times 1 = 1$ and $R_T(\mathcal{M}_2, \mathcal{M}_c) = \frac{9}{19} \times 1 + \frac{7}{19} \times 0 + \frac{3}{19} \times 0 = 0.47$.

2.4.3 Experimental Results

We report and discuss the experimental results for the Books domain. For other domains, we only show the input and output. Figure 2.7 lists all the selected attributes. The result shows two sufficiently

domain	*vocabulary (abbreviation)*
Books	title(ti),author(au),ISBN(is),keyword(kw),publisher(pu),subject(su),last name(ln), format(fm),category(cg),price(pr),first name(fn),publication date(pd)
Movies	title(ti),director(dr),actor(ac),genre(gn),format(fm),category(cg), keyword(kw),rating(rt),price(pr),studio(sd),star(st),artist(at)
Music Records	artist(at),song(sg),album(ab),title(ti),label(lb),format(fm), genre(gn),soundtrack(sr),catalog #(ct),keyword(kw),band(bn)
Automobiles	make(mk),model(md),price(pr),year(yr),type(tp),zip code(zc), mileage(ml),style(sy),color(cl),state(st),category(cg)

Figure 2.7: Vocabularies of the four domains.

domain	*output models*	P_M	R_M	P_T	R_T
Movies	$\mathcal{M}_{movie1}$ = {(ti),(dr),(fm),(rt),(pr),(sd),(kw),(ac,st),(gn,cg),(at)}	0.94	1	1	0.88
	$\mathcal{M}_{movie2}$ = {(ti),(dr),(fm),(rt),(pr),(sd),(kw),(ac,st,at),(gn),(cg)}	0.96	1	1	0.88
	$\mathcal{M}_{movie3}$ = {(ti),(dr),(fm),(rt),(pr),(sd),(kw),(ac,st,at),(gn,cg)}	1	1	1	1
Music Records	$\mathcal{M}_{music1}$ = {(sg),(lb),(fm),(at,bn),(ab,ti),(gn),(sr),(kw),(ct)}	1	1	1	1
	$\mathcal{M}_{music2}$ = {(sg),(lb),(fm),(at,bn),(ab,ti),(gn),(sr),(kw,ct)}	1	0.99	0.94	1
	$\mathcal{M}_{music3}$ = {(sg),(lb),(fm),(at,bn),(ab,ti),(gn),(sr,kw),(ct)}	1	0.99	0.94	1
	$\mathcal{M}_{music4}$ = {(sg),(lb),(fm),(at,bn),(ab,ti),(gn,sr),(kw),(ct)}	1	0.98	0.93	1
	$\mathcal{M}_{music5}$ = {(sg),(lb),(fm),(at,bn),(ab,ti),(gn,sr),(kw,ct)}	1	0.97	0.86	1
Automobiles	$\mathcal{M}_{auto}$ = {(mk),(md),(pr),(yr),(sy,tp,cg),(zc,cl),(st,ml)}	1	0.94	0.84	1

Figure 2.8: Experimental results for Movies, Music Records and Automobiles.

consistent models: $\mathcal{M}_{book1}$ = {(ti:1):.98, (is:1):.8, (kw:1):.56, (pr:1):.13, (fm:1):.13, (pd:1):.1, (pu:1):.25, (su:.61, cg:.39):.33, (au:.85, ln:.15):.98, (fn:1):.11} and $\mathcal{M}_{book2}$ = {(ti:1):.98, (is:1) :.8, (kw:1):.56, (pr:1):.13, (fm:1):.13, (pd:1):.1, (pu:1):.25, (su:.61, cg:.39):.33, (au:.88, fn:.12):.95, (ln:1):.15}.

The result successfully identifies the matchings (au, ln), (au, fn) and (su, cg). Without attribute grouping techniques (Section 2.3.1) to merge last name and first name, human experts can only consider that $\mathcal{M}_{book1}$ and $\mathcal{M}_{book2}$ both are correct schema models and thus give 1.0 precision and 1.0 recall in both model and target metrics. As stated in Section 2.3.1, attribute grouping is a different target question. Assume another specialized framework MGS_{ag} has done this task. Then the result will be $\mathcal{M}_{book}$ = {(ti:1):.98, (is:1):.8, (kw:1):.56, (pr:1):.13, (fm:1):.13, (pd:1):.1, (pu:1):.25, (su:.61, cg:.39):.33, (au:.85, [ln,fn]:.15):.98)}, which is perfectly accurate in the sense of "equivalent synonym." In addition, the parameters in the results can be used to answer the question of concept popularity (Section 2.3.1), which indicates that this model is not limited to synonym discovery.

For the other three domains: Movies, Music Records, and Automobiles, their output is summarized in Figure 2.8. The results show that our approach can identify most concepts correctly. In Movies

and Music Records, the correct schema model is returned in our output models, which are $\mathcal{M}_{movie3}$ and $\mathcal{M}_{music1}$ respectively. However, for Automobiles, we did not get the correct model. The incorrect matchings are due to the small number of observations we have. If we observe more sources, we should be able to observe some co-occurrences to remove false synonyms. For example, in the Automobile domain, the incorrect matchings (zc, cl) and (st, ml) are because we did not observe the co-occurrences of zip code and color, state and mileage. With larger observation size, we believe the result will be better.

The measurement results in Figure 2.8 show that we do need two suites of metrics because they evaluate different aspects. For instance, the model recall of $\mathcal{M}_{movie1} = 1$ means $\mathcal{M}_{movie1}$ can generate all correct instances, while the target precision of $\mathcal{M}_{movie1} = 1$ denotes the synonyms answered by $\mathcal{M}_{movie1}$ are all correct ones.

Finally, although in principle the time complexity of MGS_{ac} is exponential in terms of the number of attributes, in practice, the number of frequently used attributes within a domain is often not too many (as we have illustrated in Figure 2.1) and thus the overall execution time of our matching algorithm is quite fast, *i.e.*, within one minute (on a Pentium-III 700GHz with 128MB memory). Therefore, we believe that in practice the computation cost is likely to be acceptable for schema-matching as an off-line process.

2.5 Conclusion

This chapter explores statistical schema matching, by hypothesizing and discovering hidden models that unify input schemas. Our experience indicates high promise for moving the traditional pairwise-attribute correspondence toward a new paradigm of holistic matching of massive sources. We propose a general

statistical framework MGS, and further specialize it to develop Algorithm MGS_{ac} for finding synonym attributes. Our extensive case studies motivated our approach as well as validated its effectiveness.

However, although the MGS framework can effectively model simple matchings, it cannot find complex matchings, which generally exist across Web query interfaces (*e.g.*, author is a synonym of the grouping of last name and first name in Books domain, *i.e.*, author = {last name, first name}). To discover complex matchings, we further develop the DCM framework, as we will discuss in Chapter 3.

Chapter 3

Local Evaluation: Matching as Correlation Mining

While the MGS framework can effectively model simple matchings, it cannot find a more general type of matchings: *complex matching*. To discover complex matchings, we further develop the DCM framework [39]. Specifically, for our focus of the "deep Web," query schemas generally form complex matchings between attribute groups. In contrast to simple 1:1 matching, complex matching matches a set of *m* attributes to another set of *n* attributes, which is thus also called *m:n matching*. For instance, in the Books domain, author is a synonym of the grouping of last name and first name, *i.e.*, {author} = {first name, last name}; in the Airfares domain, {passengers} = {adults, seniors, children, infants}. Motivated by our observation that *co-occurrence* patterns across schemas often reveal the complex relationships of attributes, we develop the DCM framework by pursuing a correlation mining approach with a *local evaluation* strategy. Unlike global evaluation which evaluates an entire model, local evaluation aims at "greedily" finding individual matchings (*e.g.*, {author} = {first name, last name}) and then incrementally constructs the model.

3.1 Motivation: From Schema Matching to Correlation Mining

Our key insight is on connecting schema matching to correlation mining. Consider a typical scenario: suppose user Amy wants to book two flight tickets from city *A* to city *B*, one for her and the other for her 5-year old child. To get the best deal, she needs to query various airfare sources by filling in the Web query interfaces. For instance, in *united.com*, she fills in the query interface with from as city *A*, to as city *B* and passengers as 2. For the same query in *flyairnorth.com*, she fills in depart as city *A*, destination as city *B*, adults as 1, seniors as 0, children as 1 and infants as 0.

This scenario reveals some critical characteristics of the Web interfaces in the same domain. First, some attributes may *group* together to form a "larger" concept. For instance, the grouping of adults, seniors, children and infants denotes the number of passengers. We consider such attributes that can be grouped as *grouping attributes* or having a *grouping relationship*, denoted by putting them within braces (*e.g.*, {adults, seniors, children, infants}).

Second, different sources may use different attributes for the same concept. For instance, from and depart denote the city to leave from, and to and destination the city to go to. We consider such semantically equivalent attributes (or attribute groups) as *synonym attributes* or having a *synonym relationship*, denoted by "=" (*e.g.*, {from} = {depart}, {to} = {destination}).

Grouping attributes and synonym attributes together form *complex matchings*. In complex matching, a set of *m* attributes is matched to another set of *n* attributes, which is thus also called *m:n matching* (in contrast to the simple 1:1 matching). For instance, {adults, seniors, children, infants} = {passengers} is a 4:1 matching in the above scenario.

To tackle the complex matching problem, we exploit co-occurrence patterns to match schemas *holistically* and thus pursue a mining approach. In the holistic view of matching, all the schemas at the same time provide the co-occurrence information of attributes across many schemas, which reveals the se-

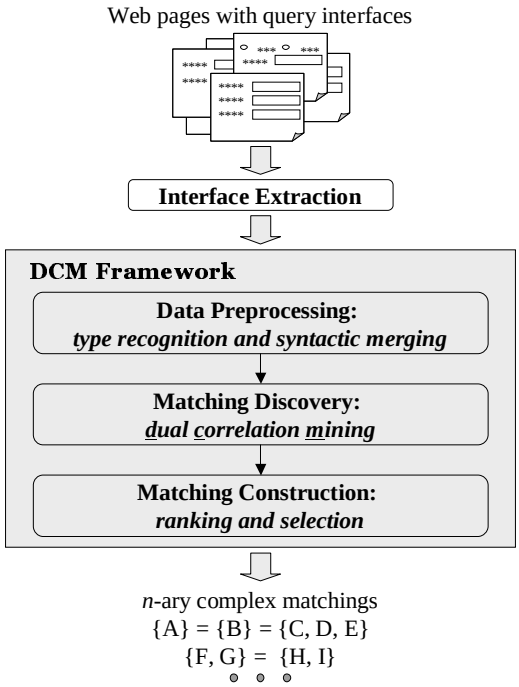


Figure 3.1: Complex matching as correlation mining.

mantics of complex matchings. (Such co-occurrence information cannot be observed when schemas are matched only in pairs.) For instance, we may observe that adults, seniors, children and infants often co-occur with each other in schemas, while they together do not co-occur with passengers. This insight enables us to discover complex matchings with a correlation mining approach. In particular, in our application, we need to handle not only positive correlations, a traditional focus, but also negative ones, which have rarely been extensively explored or applied.

By matching many schemas together, this holistic matching naturally discovers a more general type of complex matching– a matching may span more than two attribute groups. Reconsider the Amy scenario. If she tries a third airline source, *priceline.com*, she needs to fill the interface with departure city as city *A*, arrival city as city *B*, number of tickets as 2. We thus have the matching {adults, seniors, children, infants} = {passengers} = {number of tickets}, which is a 4:1:1 matching. Similarly, we have two 1:1:1 matchings {from} = {departure city} = {depart} and {to} = {arrival city} = {destination}. We

name this type of matching *n-ary complex matching*, which can be viewed as an aggregation of several binary *m*:*n* matchings.

These observations motivate us to develop a correlation mining abstraction of the schema matching problem. Specifically, given extracted schemas from Web query interfaces, we develop a streamlined process, the DCM framework, for mining complex matchings, consisting of *data preprocessing*, *matching discovery* and *matching construction*, as Figure 3.1 shows. Since the query schemas in Web interfaces are not readily minable in HTML format, before executing the DCM framework, we assume an interface extractor to extract the attribute information in the interfaces. (In this thesis, we will also address the impact of errors made by the automatic interface extractor on our matching algorithm in Chapter 4.) Given extracted raw schema data, we first preprocess schemas to make them ready for mining as the data preprocessing step (Section 3.4). Next, the matching discovery step, the core of the DCM framework, explores a *dual correlation mining* algorithm to discover *n*-ary complex matchings, which first mines potential attribute groups as positive correlations and then potential complex matchings as negative correlations (Section 3.2.1). Finally, matching construction ranks and then selects the most confident and consistent matchings from the mining result (Section 3.2.2). Meanwhile, in the heart of correlation mining, we need to choose an appropriate correlation measure (Section 3.3).

3.2 Complex matching as correlation mining

We view a schema as a *transaction*, a conventional abstraction in association and correlation mining. In data mining, a transaction is a set of items; correspondingly, in schema matching, we consider a schema as a set of *attribute entities*. An attribute entity contains attribute name, type and domain (*i.e.*, instance values). Before mining, the data preparation step (Section 3.4) finds syntactically similar entities among schemas. After that, each attribute entity is assigned a unique *attribute identifier*. While the mining is

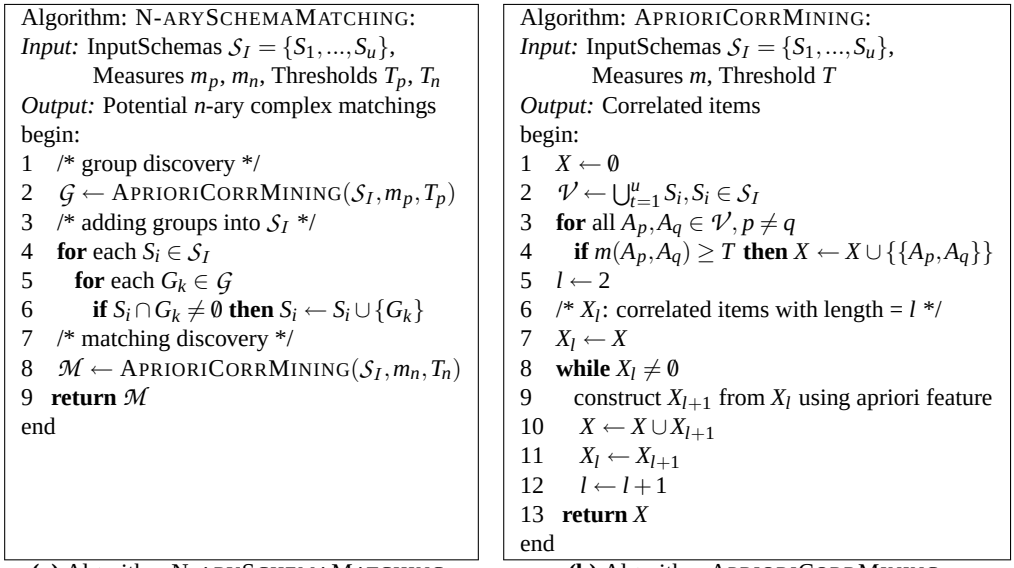
over the attribute entities, for simplicity of illustration, we use the attribute name of each entity, after cleaning, as the attribute identifier.

Formally, we consider the schema matching problem as: *Given the input as a set of schemas* $\mathcal{S}_I = \{S_1, ..., S_u\}$ *in the same domain, where each schema* S_i *is a transaction of attribute identifiers, find all the matchings* $\mathcal{M} = \{M_1, ..., M_v\}$. *Each* M_j *is an n-ary complex matching* $G_{j_1} = G_{j_2} = ... = G_{j_w}$, *where each* G_{j_k} *is an attribute group and* $G_{j_k} \subseteq \bigcup_{i=1}^{u} S_i$. Semantically, each M_j should represent the synonym relationship of attribute groups G_{j_1},..., G_{j_w} and each G_{j_k} should represent the grouping relationship of attributes in G_{j_k}.

Motivated by our observations on the schema data (Section 3.1), we develop a correlation mining algorithm, with respect to the above abstraction (Figure 3.1), consists of *dual correlation mining* and *matching construction*. We will elaborate these two steps in Section 3.2.1 and Section 3.2.2 respectively.

Briefly, the dual correlation mining has two sub-steps. First, *group discovery*: We mine *positively correlated attributes* to form potential attribute groups. A potential group may not be eventually useful for matching; only the ones having a synonym relationship (*i.e.*, negative correlation) with other groups can survive. For instance, if all sources use last name, first name, and not author, then the potential group {last name, first name} is not useful because there is no matching (to author) needed. Second, *matching discovery*: Given the potential groups (including singleton ones), we mine *negatively correlated attribute groups* to form potential *n*-ary complex matchings. A potential matching may not be considered as correct due to the existence of conflicts among matchings.

After group discovery, we need to add the discovered groups into the input schemas $\mathcal{S}_I$ to mine negative correlations among groups. (A single attribute is viewed as a group with only one attribute.) Specifically, an attribute group is added into a schema if that schema contains any attribute in the group. For instance, if we discover that last name and first name have a grouping relationship, we consider



Algorithm: N-ARYSCHEMAMATCHING:
Input: InputSchemas $\mathcal{S}_I = \{S_1, ..., S_u\}$,
Measures m_p, m_n, Thresholds T_p, T_n
Output: Potential n-ary complex matchings
begin:
1 /* group discovery */
2 $\mathcal{G} \leftarrow$ APRIORICORRMINING$(\mathcal{S}_I, m_p, T_p)$
3 /* adding groups into $\mathcal{S}_I$ */
4 **for** each $S_i \in \mathcal{S}_I$
5 **for** each $G_k \in \mathcal{G}$
6 **if** $S_i \cap G_k \neq \emptyset$ **then** $S_i \leftarrow S_i \cup \{G_k\}$
7 /* matching discovery */
8 $\mathcal{M} \leftarrow$ APRIORICORRMINING$(\mathcal{S}_I, m_n, T_n)$
9 **return** $\mathcal{M}$
end

Algorithm: APRIORICORRMINING:
Input: InputSchemas $\mathcal{S}_I = \{S_1, ..., S_u\}$,
Measures m, Threshold T
Output: Correlated items
begin:
1 $X \leftarrow \emptyset$
2 $\mathcal{V} \leftarrow \bigcup_{t=1}^{u} S_i, S_i \in \mathcal{S}_I$
3 **for** all $A_p, A_q \in \mathcal{V}, p \neq q$
4 **if** $m(A_p, A_q) \geq T$ **then** $X \leftarrow X \cup \{\{A_p, A_q\}\}$
5 $l \leftarrow 2$
6 /* X_l: correlated items with length = l */
7 $X_l \leftarrow X$
8 **while** $X_l \neq \emptyset$
9 construct X_{l+1} from X_l using apriori feature
10 $X \leftarrow X \cup X_{l+1}$
11 $X_l \leftarrow X_{l+1}$
12 $l \leftarrow l + 1$
13 **return** X
end

(a) Algorithm N-ARYSCHEMAMATCHING. **(b)** Algorithm APRIORICORRMINING.

Figure 3.2: Algorithms for Mining Complex Matchings.

{last name, first name} as an attribute group, denoted by G_{lf} for simplicity, and add it to any schema containing either last name or first name, or both. The intuition is that although a schema may not contain the entire group, it still partially covers the concept that the group denotes and thus should be counted in matching discovery for that concept. Note that we do not remove singleton groups {last name} and {first name} when adding G_{lf}, because G_{lf} is only a potential group and may not survive in matching discovery.

The matching construction also has two sub-steps: First, *matching ranking*: To solve the conflicts, we develop a ranking strategy to rank the confidence of each matching candidate discovered by the dual correlation mining phase. Second, *matching selection*: We further develop a selection strategy to select the most confident and consistent matchings from the mining result according to the rankings.

3.2.1 Matching Discovery: Dual Correlation Mining

While group discovery works on individual attributes and matching discovery on attribute groups, they can share the same mining process. We use the term – *items* – to represent both attributes and groups in the following discussion of mining algorithms.

Correlation mining, at the heart, requires a measure to gauge correlation of a set of n items; our observation indicates pairwise correlations among these n items. Specifically, for n groups forming synonyms, any two groups should be negatively correlated, since they both are synonyms by themselves (*e.g.*, in the matching $\{\mathsf{destination}\} = \{\mathsf{to}\} = \{\mathsf{arrival\ city}\}$, negative correlations exist between any two groups). We have a similar observation on the attributes with grouping relationships. Motivated by such observations, we design the measure as:

$$C_{min}(\{A_1,...,A_n\},m) = \min m(A_i,A_j), \forall i \neq j, \tag{3.1}$$

where m is some correlation measure for two items (*e.g.*, the measures surveyed in [63]). That is, we define C_{min} as the minimal value of the pairwise evaluation, thus requiring all pairs to meet this minimal "strength."

C_{min} has several advantages: First, it satisfies the "apriori" feature and thus enables the design of an efficient algorithm. In correlation mining, the measure for qualification should have a desirable "apriori" property (*i.e.*, downward closure), to develop an efficient algorithm. (In contrast, a measure for ranking should not have this "apriori" feature, as Section 3.2.2 will discuss.) C_{min} satisfies the "apriori" feature since given any item set $\mathcal{A}$ and its subset $\mathcal{A}^*$, we have $C_{min}(\mathcal{A}, m) \leq C_{min}(\mathcal{A}^*, m)$ because the minimum of a larger set (*e.g.*, min($\{1,3,5\}$)) cannot be higher than its subset (*e.g.*, min($\{3,5\}$)). Second, C_{min}

can incorporate any measure m for two items and thus can accommodate different tasks (*e.g.*, mining positive and negative correlations) and be customized to achieve good mining quality.

Leveraging the "apriori" feature of C_{min}, we develop Algorithm APRIORICORRMINING (Figure 3.2) for discovering complex matchings, in the spirit of the classic Apriori algorithm for association mining [1]. That is, we find all the correlated items with length $l+1$ based on the ones with length l.

With C_{min}, we can directly define positively correlated attributes in group discovery and negatively correlated attribute groups in matching discovery. A set of attributes $\{A_1, ..., A_n\}$ is *positively correlated attributes*, denoted by *PC*, if $C_{min}(\{A_1, ..., A_n\}, m_p) \geq T_p$, where m_p is a measure for positive correlation and T_p is a given threshold. Similarly, a set of attribute groups $\{G_1, ..., G_m\}$ is *negatively correlated attribute groups*, denoted by *NC*, if $C_{min}(\{G_1, ..., G_m\}, m_n) \geq T_n$, where m_n is a measure for negative correlation and T_n is another given threshold.

Algorithm N-ARYSCHEMAMATCHING shows the pseudo code of the complex matching discovery (Figure 3.2). Line 2 (group discovery) calls APRIORICORRMINING to mine PC. Lines 3-6 add the discovered groups into $\mathcal{S}_I$. Line 8 (matching discovery) calls APRIORICORRMINING to mine NC. Similar to [1], the time complexity of N-ARYSCHEMAMATCHING is exponential with respect to the number of attributes. But in practice, the execution is quite fast since correlations exist among semantically related attributes, which is far less than arbitrary combination of all attributes.

3.2.2 Matching Construction: Majority-based Ranking and Constraint-based Selection

After the matching discovery step, we need to develop ranking and selection strategies for the matching construction step. We notice that the matching discovery step can discover true semantic matchings and, as expected, also false ones due to the existence of coincidental correlations. For instance, in the Books domain, the mining result may have both {author} = {first name, last name}, denoted by M_1

Algorithm: MATCHINGSELECTION:
Input: Potential matchings $\mathcal{M} = \{M_1, ..., M_v\}$,
Measure m_n
Output: Selected matchings
begin:
1 $\mathcal{R} \leftarrow \emptyset$ /* selected n-ary complex matchings */
2 **while** $\mathcal{M} \neq \emptyset$
3 /* select the matching ranked the highest */
4 $M_t \leftarrow$ GETMATCHINGRANKFIRST($\mathcal{M}$, m_n)
5 $\mathcal{R} \leftarrow \mathcal{R} \cup \{M_t\}$
6 **for** each $M_j \in \mathcal{M}$
7 /* remove the conflicting part */
8 $M_j \leftarrow M_j - M_t$
9 /* delete M_j if it contains no matching */
10 **if** $|M_j| < 2$ **then** $\mathcal{M} \leftarrow \mathcal{M} - \{M_j\}$
11 **return** $\mathcal{R}$
end

(a) Algorithm MATCHINGSELECTION.

Algorithm: GETMATCHINGRANKFIRST:
Input: Potential matchings $\mathcal{M} = \{M_1, ..., M_v\}$,
Measure m_n
Output: The matching with the highest ranking
begin:
1 $M_t \leftarrow M_1$
2 **for** each $M_j \in \mathcal{M}, 2 \leq j \leq v$
3 **if** $s(M_j, m_n) > s(M_t, m_n)$ **then**
4 $M_t \leftarrow M_j$
5 **if** $s(M_j, m_n) = s(M_t, m_n)$ and $M_j \succeq M_t$ **then**
6 $M_t \leftarrow M_j$
7 **return** M_t
end

(b) Algorithm GETMATCHINGRANKFIRST.

Figure 3.3: Algorithm MATCHINGSELECTION.

and $\{\text{subject}\} = \{\text{first name, last name}\}$, denoted by M_2. We can see M_1 is correct, while M_2 is not. The reason for having the false matching M_2 is that in the schema data, it happens that subject rarely co-occurs with first name and last name.

The existence of false matchings may cause matching conflicts. For instance, M_1 and M_2 conflict in that if one of them is correct, the other one will not. Otherwise, we get a wrong matching $\{\text{author}\} = \{\text{subject}\}$ by the transitivity of the synonym relationship. Since our mining algorithm does not discover $\{\text{author}\} = \{\text{subject}\}$, M_1 and M_2 cannot be both correct.

Leveraging the conflicts, we select the most confident and consistent matchings to remove the false ones. Intuitively, between conflicting matchings, we want to select the more negatively correlated one because it indicates higher confidence to be synonyms. For example, our experiment shows that, as M_2 is coincidental, it is indeed that $m_n(M_1) > m_n(M_2)$, and thus we select M_1 and remove M_2. With larger data size, semantically correct matching is more likely to be the winner. The reason is that, with larger sampling size, the correct matchings are still negatively correlated while the false ones will remain coincidental and not as strong.

Before presenting the selection algorithm, we need to develop a strategy for *ranking* the discovered matchings. That is, for any n-ary complex matching M_j: $G_{j_1} = G_{j_2} = ... = G_{j_w}$, we have a score function $s(M_j, m_n)$ to evaluate M_j under measure m_n.

Section 3.2.1 discussed a measure for "qualifying" candidates. We now need to develop another "ranking" measure as the score function. Since ranking and qualification are different problems and thus require different properties, we cannot apply the measure C_{min} (Equation 3.1) for ranking. Specifically, the goal of qualification is to ensure the correlations passing some threshold. It is desirable for the measure to support downward closure to enable an "apriori" algorithm. In contrast, the goal of ranking is to compare the strength of correlations. The downward closure enforces, by definition, that a larger item set is always less correlated than its subsets, which is inappropriate for ranking correlations of different sizes. Hence, we develop another measure C_{max}, the maximal m_n value among pairs of groups in a matching, as the score function s. Formally,

$$C_{max}(M_j, m_n) = \max m_n(G_{j_r}, G_{j_t}), \forall G_{j_r}, G_{j_t}, j_r \neq j_t. \tag{3.2}$$

It is possible to get ties if only considering the C_{max} value; we thus develop a natural strategy for tie breaking. We take a "top-k" approach so that s in fact is a set of sorted scores. Specifically, given matchings M_j and M_k, if $C_{max}(M_j, m_n) = C_{max}(M_k, m_n)$, we further compare their second highest m_n values to break the tie. If the second highest values are also equal, we compare the third highest ones and so on, until breaking the tie.

If two matchings are still tied after the "top-k" comparison, we choose the one with richer semantic information. We consider matching M_j to *semantically subsume* matching M_k, denoted by $M_j \succeq M_k$, if all the semantic relationships in M_k are covered in M_j. For instance, $\{\text{arrival city}\} = \{\text{destination}\} = \{\text{to}\} \succeq \{\text{arrival city}\} = \{\text{destination}\}$ because the synonym relationship in the second matching is subsumed in

the first one. Also, {author} = {first name, last name} $\succeq$ {author} = {first name} because the synonym relationship in the second matching is part of the first.

Combining the score function and semantic subsumption, we rank matchings with the following rules: 1) If $s(M_j, m_n) > s(M_k, m_n)$, M_j is ranked higher than M_k. 2) If $s(M_j, m_n) = s(M_k, m_n)$ and $M_j \succeq M_k$, M_j is ranked higher than M_k. 3) Otherwise, we rank M_j and M_k arbitrarily. Algorithm GETMATCHINGRANKFIRST (Figure 3.3) illustrates the pseudo code of choosing the highest ranked matching with this strategy.

Algorithm MATCHINGSELECTION shows the selection algorithm. We apply a greedy selection strategy by choosing the highest ranked matching, M_t, in each iteration. After choosing M_t, we remove the inconsistent parts in remaining matchings (lines 6 - 10). The final output is the selected n-ary complex matchings without conflict. Note that we need to do the ranking in each iteration instead of sorting all the matchings in the beginning because after removing the conflicting parts, the ranking may change. The time complexity of Algorithm MATCHINGSELECTION is $O(v^2)$, where v is the number of matchings in $\mathcal{M}$.

Example 14: Assume running N-ARYSCHEMAMATCHING in the Books domain finds matchings $\mathcal{M}$ as (matchings are followed by their scores):

M_1: {author} = {last name, first name}, 0.95

M_2: {author} = {last name}, 0.95

M_3: {subject} = {category}, 0.92

M_4: {author} = {first name}, 0.90

M_5: {subject} = {last name, first name} , 0.88

M_6: {subject} = {last name}, 0.88 and

M_7: {subject} = {first name}, 0.86.

	A_p	$\neg A_p$	
A_q	f_{11}	f_{10}	f_{1+}
$\neg A_q$	f_{01}	f_{00}	f_{0+}
	f_{+1}	f_{+0}	f_{++}

Figure 3.4: Contingency table for test of correlation.

In the first iteration, M_1 is ranked the highest and thus selected. In particular, although $s(M_1, m_n) = s(M_2, m_n)$, M_1 is ranked higher since $M_1 \succeq M_2$. Now we remove the conflicting parts of the other matchings. For instance, M_2 conflicts with M_1 on author. After removing author, M_2 only contains one attribute and is not a matching any more. So we remove M_2 from $\mathcal{M}$. Similarly, M_4 and M_5 are also removed. The remaining matchings are M_3, M_6 and M_7. In the second iteration, M_3 is ranked the highest and thus also selected. M_6 and M_7 are removed because they conflict with M_3. Now $\mathcal{M}$ is empty and the algorithm stops. The final output is thus M_1 and M_3.

3.3 Correlation Measure

In this section, we discuss the positive measure m_p and the negative measure m_n, used as the component of C_{min} (Equation 3.1) for positive and negative correlation mining respectively in Algorithm N-ARYSCHEMAMATCHING (Section 3.2).

As discussed in [63], a correlation measure by definition is a testing on the *contingency table*. Specifically, given a set of schemas and two specified attributes A_p and A_q, there are four possible combinations of A_p and A_q in one schema S_i: A_p,A_q are co-present in S_i, only A_p presents in S_i, only A_q presents in S_i, and A_p,A_q are co-absent in S_i. The *contingency table* [14] of A_p and A_q contains the number of occurrences of each situation, as Figure 3.4 shows. In particular, f_{11} corresponds to the number of co-presence of A_p and A_q; f_{10}, f_{01} and f_{00} are denoted similarly. f_{+1} is the sum of f_{11} and f_{01}; f_{+0}, f_{0+} and f_{1+} are denoted similarly. f_{++} is the sum of f_{11}, f_{10}, f_{01} and f_{00}.

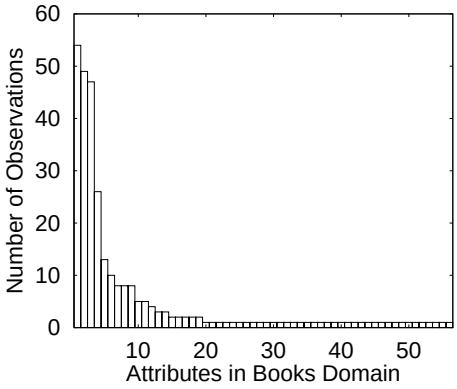


Figure 3.5: Attribute frequencies in the Books domain.

The design of a correlation measure is often empirical. To our knowledge, there is no good correlation measure universally agreed upon [63]. For our matching task, in principle *any* measure can be applied. However, since the semantic correctness of the mining result is of special importance for the schema matching task, we especially care about the ability of the measures to differentiate various correlation situations, especially the subtlety of negative correlations, which has not been extensively studied before.

We first identify the quality requirements of measures, which are imperative for schema matching, based on the characteristics of Web query interfaces. Specifically, we observe that, in Web interfaces, attribute frequencies are extremely non-uniform, similar to the use of English words, showing some Zipf-like distribution. For instance, Figure 3.5 shows the attribute frequencies in the Books domain: First, the non-frequent attributes result in the sparseness of the schema data (*e.g.*, there are over 50 attributes in the Books domain, but each schema only has 5 on average). Second, many attributes are rarely used, occurring only once in the schemas. Third, there exist some highly frequent attributes, occurring in almost every schema.

These three observations indicate that, as the quality requirements, the chosen measures should be robust against the following problems: *sparseness problem* for both positive and negative correlations,

	A_p	$\neg A_p$	
A_q	5	5	10
$\neg A_q$	5	85	90
	10	90	100

(a1) Example of sparseness problem with measure *Lift*: Less positive correlation but a higher *Lift* = 17.

	A_p	$\neg A_p$	
A_q	1	49	50
$\neg A_q$	1	1	2
	2	50	52

(b1) Example of rare attribute problem with measure *Jaccard*: A_p as rare attribute and *Jaccard* = 0.02.

	A_p	$\neg A_p$	
A_q	81	9	90
$\neg A_q$	9	1	10
	90	10	100

(c1) Example of frequent attribute problem with measure *Jaccard*: A_p and A_q are independent but a higher *Jaccard* = 0.82.

	A_p	$\neg A_p$	
A_q	55	20	75
$\neg A_q$	20	5	25
	75	25	100

(a2) Example of sparseness problem with measure *Lift*: More positive correlation but a lower *Lift* = 0.69.

	A_p	$\neg A_p$	
A_q	1	25	26
$\neg A_q$	25	1	26
	26	26	52

(b2) Example of rare attribute problem with measure *Jaccard*: no rare attribute and *Jaccard* = 0.02.

	A_p	$\neg A_p$	
A_q	8	1	9
$\neg A_q$	1	90	91
	9	91	100

(c2) Example of frequent attribute problem with measure *Jaccard*: A_p and A_q are positively correlated but a lower *Jaccard* = 0.8.

Figure 3.6: Examples of the three problems.

rare attribute problem for negative correlations, and *frequent attribute problem* for positive correlations. In this section, we discuss each of them in detail.

The Sparseness Problem

In schema matching, it is more interesting to measure whether attributes are often co-present (*i.e.*, f_{11}) or cross-present (*i.e.*, f_{10} and f_{01}) than whether they are co-absent (*i.e.*, f_{00}). Many correlation measures, such as χ^2 and *Lift*, include the count of co-absence in their formulas. This may not be good for our matching task, because the sparseness of schema data may "exaggerate" the effect of co-absence. This problem has also been noticed by recent correlation mining work such as [63, 56, 48]. In [63], the authors use the *null invariance* property to evaluate whether a measure is sensitive to co-absence. The measures for our matching task should satisfy this null invariance property.

Example 15: Figure 3.6(a) illustrates the sparseness problem with an example. In this example, we choose a common measure: the *Lift* (i.e, $Lift = \frac{f_{00} f_{11}}{f_{10} f_{01}}$). (Other measures considering f_{00} have similar behavior.) The value of *Lift* is between 0 to $+\infty$. $Lift = 1$ means independent attributes, $Lift > 1$ positive correlation and $Lift < 1$ negative correlation. Figure 3.6(a) shows that *Lift* may give a higher value to less positively correlated attributes. In the scenario of schema matching, the table in Figure 3.6(a2)

should be more positively correlated than the one in Figure 3.6(a1) because in Figure 3.6(a2), the co-presence (f_{11}) is more frequently observed than the cross-presence (either f_{10} or f_{01}), while in Figure 3.6(a1), the co-presence has the same number of observations as the cross-presence. However, *Lift* cannot reflect such preference. In particular, Figure 3.6(a1) gets a much higher *Lift* and Figure 3.6(a2) is even evaluated as not positively correlated. A similar example can also be found for negative correlation with *Lift*. The reason *Lift* gives an inappropriate answer is that f_{00} incorrectly affects the result.

We explored the 21 measures in [63] and the χ^2 measure in [12]. Most of these measures (including χ^2 and *Lift*) suffer the sparseness problem. That is, they consider both co-presence and co-absence in the evaluation and thus do not satisfy the null invariance property. The only three measures supporting the null invariance property are *Confidence*, *Jaccard* and *Cosine*.

The Rare Attribute Problem for Negative Correlation

Although *Confidence*, *Jaccard* and *Cosine* satisfy the null invariance property, they are not robust for the rare attribute problem, when considering negative correlations. Specifically, the rare attribute problem can be stated as when either A_p or A_q is rarely observed, the measure should not consider A_p and A_q as highly negatively correlated because the number of observations is not convincing to make such judgement. For instance, the *Jaccard* (*i.e.*, *Jaccard* = $\frac{f_{11}}{f_{11}+f_{10}+f_{01}}$) measure will stay unchanged when both f_{11} and $f_{10} + f_{01}$ are fixed. Therefore, to some degree, *Jaccard* cannot differentiate the subtlety of correlations (*e.g.*, $f_{10} = 49$, $f_{01} = 1$ and $f_{10} = 25$, $f_{01} = 25$), as Example 16 illustrates. Other measures such as *Confidence* and *Cosine* have a similar problem. This problem is not so critical for positive correlation, since attributes with strong positive correlations cannot be rare.

Example 16: Figure 3.6(b) illustrates the rare attribute problem. In this example, we choose a common measure: the *Jaccard*. The value of *Jaccard* is between 0 to 1. *Jaccard* close to 0 means negative correlation and *Jaccard* close to 1 positive correlation. Figure 3.6(b) shows that *Jaccard* may not be

able to distinguish the situations of rare attribute. In particular, Jaccard considers the situations in Figure 3.6(b1) and Figure 3.6(b2) as the same. But Figure 3.6(b2) is more negatively correlated than Figure 3.6(b1) because A_p in Figure 3.6(b1) is more like a rare event than a true negative correlation.

To differentiate the subtlety of negative correlations, we develop a new measure, H-measure (Equation 3.3), as the negative correlation m_n. The value of H is in the range from 0 to 1. An H value close to 0 denotes a high degree of positive correlation; an H value close to 1 denotes a high degree of negative correlation.

$$m_n(A_p, A_q) = H(A_p, A_q) = \frac{f_{01} f_{10}}{f_{+1} f_{1+}}. \tag{3.3}$$

H-measure satisfies the quality requirements: On the one hand, similar to *Jaccard*, *Cosine* and *Confidence*, H-measure satisfies the null invariance property and thus avoids the sparseness problem by ignoring f_{00}. On the other hand, by multiplying the individual effect of f_{01} (*i.e.*, $\frac{f_{01}}{f_{+1}}$) and f_{10} (*i.e.*, $\frac{f_{10}}{f_{1+}}$), H-measure is more capable of reflecting subtle variation of negative correlations.

The Frequent Attribute Problem for Positive Correlation

For positive correlations, we find that *Confidence*, *Jaccard*, *Cosine* and H-measure are not quite different in discovering attribute groups. However, all of them suffer from the frequent attribute problem. This problem seems to be essential for these measures: Although they avoid the sparseness problem by ignoring f_{00}, as the cost, they lose the ability to differentiate highly frequent attributes from really correlated ones. Specifically, highly frequent attributes may co-occur in most schemas just because they are so frequently used, not because they have grouping relationship (*e.g.*, In the Books domain, isbn and title are often co-present because they are both very frequently used). This phenomenon may generate uninteresting groups (*i.e.*, *false positives*) in group discovery.

Example 17: Figure 3.6(c) illustrates the frequent attribute problem with an example, where we still use *Jaccard* as the measure. Figure 3.6(c) shows that *Jaccard* may give a higher value to independent attributes. In Figure 3.6(c1), A_p and A_q are independent and both of them have the probabilities 0.9 to be observed, while in Figure 3.6(c2), A_p and A_q are really positively correlated. However, *Jaccard* considers Figure 3.6(c1) as more positively correlated than Figure 3.6(c2). In our matching task, a measure should not give a high value for frequently observed but independent attributes.

The characteristic of false groupings is that the f_{11} value is very high (since both attributes are frequent). Based on this characteristic, we add another measure $\frac{f_{11}}{f_{++}}$ in m_p to filter out false groupings. Specifically, we define the positive correlation measure m_p as:

$$m_p(A_p,A_q) = \begin{cases} 1-H(A_p,A_q), & \frac{f_{11}}{f_{++}} < T_d \\ 0, & otherwise, \end{cases} \tag{3.4}$$

where T_d is a threshold to filter out false groupings. To be consistent with m_n, we also use the H-measure in m_p.

3.4 Data Preparation

As input of the DCM framework, we assume an interface extractor (Figure 3.1) has extracted attribute information from Web interfaces in HTML formats. (Chapter 4 will discuss the incorporation of an automatic interface extractor [72].) The extracted raw schemas contain many syntactic variations around the "core" concept (*e.g.*, title) and thus are not readily minable. We thus perform a data preprocessing step to make schemas ready for mining. The data preprocessing step consists of *attribute normalization*, *type recognition* and *syntactic merging*. To begin with, given extracted schema data, we perform some standard normalization on the extracted names and domain values. We first stem attribute names and

domain values using the standard Porter stemming algorithm [59]. Next, we normalize irregular nouns and verbs (*e.g.*, "children" to "child," "colour" to "color"). Last, we remove common stop words by a manually built stop word list, which contains words common in English, in Web search (*e.g.*, "search", "page"), and in the respective domain of interest (*e.g.*, "book", "movie").

We then perform type recognition to identify attribute types. As Section 3.4.1 discusses, type information helps to identify homonyms (*i.e.*, two attributes may have the same name but different types) and constrain syntactic merging and correlation-based matching (*i.e.*, only attributes with compatible types can be merged or matched). Since the type information is not declared in Web interfaces, we develop a *type recognizer* to recognize types from domain values.

Finally, we merge attribute entities by measuring the syntactic similarity of attribute names and domain values (*e.g.*, we merge "title of book" to "title" by name similarity). It is a common data cleaning technique to merge syntactically similar entities by using a linguistic approach. Section 3.4.2 discusses our merging strategy.

3.4.1 Type Recognition

While attribute names can distinguish different attribute entities, the names alone sometimes lead to the problem of homonyms (*i.e.*, the same name with different meanings) – we address this problem by distinguishing entities by both names and types. For instance, the attribute name departing in the Airfares domain may have two meanings: a datetime type as departing date, or a string type as departing city. With type recognition, we can recognize that there are two different types of departing: departing (datetime) and departing (string), which indicate two attribute entities.

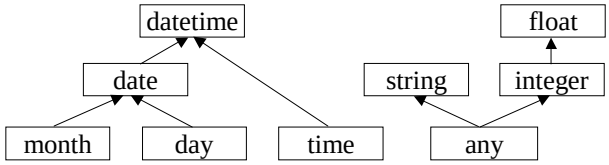


Figure 3.7: The compatibility of types.

In general, type information, as a constraint, can help the subsequent steps of syntactic merging and correlation-based matching. In particular, if the types of two attributes are not compatible, we consider they denote different attribute entities and thus neither merge them nor match them.

Since type information is not explicitly declared in Web interfaces, we develop a *type recognizer* to recognize types from domain values of attribute entities. For example, a list of integer values denotes an integer type. In the current implementation, type recognition supports the following types: any, string, integer, float, month, day, date, time and datetime. (An attribute with only an input box is given an any type since the input box can accept data with different types such as string or integer.) Two types are *compatible* if one can subsume another (*i.e.*, the *is-a* relationship). For instance, date and datetime are compatible since date subsumes datetime. Figure 3.7 lists the compatibility of all the types in our implementation.

3.4.2 Syntactic Merging

We clean the schemas by merging syntactically similar attribute entities, a common data cleaning technique to identify unique entities [19]. Specifically, we develop *name-based merging* and *domain-based merging* by measuring the syntactic similarity of attribute names and domains respectively. Syntactic merging increases the observations of attribute entities, which can improve the effect of correlation evaluation.

Name-based Merging: We merge two attribute entities if they have similar names. We observe that the majority of deep Web sources are consistent on some concise "core" attribute names (*e.g.*,

"title") and others are variations of the core ones (*e.g.*, "title of book"). Therefore, we consider attribute A_p is *name-similar* to attribute A_q if A_p's name $\supseteq$ A_q's name (*i.e.*, A_p is a variation of A_q) and A_q is more frequently used than A_p (*i.e.*, A_q is the majority). This frequency-based strategy helps avoid false positives. For instance, in the Books domain, last name is not merged to name because last name is more popular than name and thus we consider them as different entities.

Domain-based Merging: We then merge two attribute entities if they have similar domain values. In particular, we only consider attributes with string types, since it is unclear how useful it is to measure the domain similarity of other types. For instance, in the Airfares domain, the integer values of passengers and connections are quite similar, although they denote different meanings.

We view domain values as a bag of words (*i.e.*, counting the word frequencies). We name such a bag *aggregate values*, denoted as V_A for attribute A. Given a word w, we denote $V_A(w)$ as the frequency of w in V_A. The domain similarity of attributes A_p and A_q is thus the similarity of V_{A_p} and V_{A_q}. In principle, any reasonable similarity function is applicable here. In particular, we choose $sim(A_p, A_q) = \frac{\forall w \in V_{A_p} \cap V_{A_q}, V_{A_p}(w) + V_{A_q}(w)}{\forall w \in V_{A_p} \cup V_{A_q}, V_{A_p}(w) + V_{A_q}(w)}$.

The above three steps, form extraction, type recognition and syntactic merging, clean the schema data as transactions to be mined. More detailed discussion about these data cleaning steps can be found at the extended report [38].

3.5 Experiments

We collected 447 deep Web sources in 8 popular domains in the format of raw Web pages as our testbed, where each domain has about 20-70 deep Web sources. This dataset is available as the TEL-8 dataset in the UIUC Web Integration Repository [17].

In the experiment, we assume a perfect form extractor to extract all the interfaces in the TEL-8 dataset into query capabilities by manually doing the form extraction step. The reason we do not apply the work in [72] is that we want to isolate the mining process to study and thus fairly evaluate the matching performance. (Chapter 4 will systematically study the impact of form extractor to matching performance.) After extracting the raw data, we do the data cleaning according to the process explained in Section 3.4. Then, we run the correlation mining algorithm on the prepared data in each domain. Also, in the results, we use attribute name and type together as the attribute identifier for an attribute entity since we incorporate type recognition in data preparation to identify homonyms (Section 3.4).

To evaluate the performance of the algorithms we have developed in this chapter, we conduct four sets of experiment on the TEL-8 dataset. First, we test our approach on the TEL-8 dataset and the result shows good *target accuracy*. We also evaluate the effectiveness of the matching selection algorithm and the data preprocessing step. Last, we compare the H-measure with other measures on the TEL-8 dataset and the result shows that H-measure outperforms the others in most cases.

3.5.1 Metrics

We compare experimentally discovered matchings, denoted by $\mathcal{M}_h$, with correct matchings written by human experts, denoted by $\mathcal{M}_c$. In particular, we adopt the *target accuracy*, a metric initially developed in the MGS framework (Chapter 2), by customizing the *target questions* to the complex matching scenario. The idea of the target accuracy is to measure how accurately that the discovered matchings answer the target questions. Specifically, for our complex matching task, we consider the target question as, given any attribute, to find its synonyms (*i.e.*, word with exactly the same meaning as another word, *e.g.*, subject is a synonym of category in the Books domain), hyponyms (*i.e.*, word of more specific

meaning, *e.g.*, last name is a hyponym of author) and hypernyms (*i.e.*, word with a broader meaning, e.g, author is a hypernym of last name).

It is quite complicated to use different measures for different semantic relationships. We therefore report an aggregate measure for simplicity and, at the same time, still reflect semantic differences. For our discussion here, we name synonym, hyponym and hypernym together as *closenym* – meaning that they all denote some degrees of closeness in semantic meanings. Our target question now is to ask the set of closenyms of a given attribute.

Example 18: For instance, for matching {A} = {B, C}, the closenym sets of attributes A, B, C are {B, C}, {A}, {A} respectively. In particular, the closenym sets of B does not have C since B and C only have grouping relationship. In contrast, for matching {A} = {B} = {C}, the closenym sets of attributes A, B, C are {B, C}, {A, C}, {A, C} respectively. We can see that the difference of matchings can be reflected in the corresponding closenym sets.

Except for this difference in target question, we use the same metric of target accuracy as in the MGS framework. Specifically, we assume a "random querier" to ask for closenym set of each attribute according to its frequency. The answer to each question is closenym set of the queried attribute in discovered matchings. We define $Cls(A_j|\mathcal{M})$ as the closenym set of attribute A_j. Given $\mathcal{M}_c$ and $\mathcal{M}_h$, the precision and recall of the closenym sets of attribute A_j are:

$P_{A_j}(\mathcal{M}_h, \mathcal{M}_c) = \frac{|Cls(A_j|\mathcal{M}_c) \cap Cls(A_j|\mathcal{M}_h)|}{|Cls(A_j|\mathcal{M}_h)|}$ and

$R_{A_j}(\mathcal{M}_h, \mathcal{M}_c) = \frac{|Cls(A_j|\mathcal{M}_c) \cap Cls(A_j|\mathcal{M}_h)|}{|Cls(A_j|\mathcal{M}_c)|}$.

Since more frequently used attributes have higher probabilities to be asked in this "random querier," we compute the weighted average of all the P_{A_j}'s and R_{A_j}'s as the *target precision* and *target recall*. The weight is assigned as $\frac{O_j}{\sum O_k}$, where O_j is the frequency of attribute A_j in the dataset (*i.e.*, its number of

Step	*Value of*	*Result*	C_{min}	C_{max}
group discovery	$\mathcal{G}$	G_1 = {last name (unknown), first name (any)}	0.94	
		G_2 = {title (any), keyword (any)}	0.93	
		G_3 = {last name (any), title (any)}	0.91	
		G_4 = {first name (any), catalog (any)}	0.90	
		G_5 = {first name (any), keyword (any)}	0.87	
matching discovery	$\mathcal{M}$	M_1: {author (any)} = {last name (any), first name (any)}	0.87	0.87
		M_2: {author (any)} = {last name (any)}	0.87	0.87
		M_3: {subject (string)} = {category (string)}	0.83	0.83
		M_4: {author (any)} = {last name (any), catalog (any)}	0.82	0.82
		M_5: {author (any)} = {first name (any)}	0.82	0.82
		M_6: {category (string)} = {publisher (string)}	0.76	0.76
matching selection	$\mathcal{R}$	R_1: {author (any)} = {last name (any), first name (any)}		0.87
		R_2: {subject (string)} = {category (string)}		0.83

Figure 3.8: Running Algorithms N-ARYSCHEMAMATCHING and MATCHINGSELECTION on the Books domain.

Domain	*Final Output After Matching Selection*	*Correct?*
Airfares	{destination (string)} = {to (string)} = {arrival city (string)}	Y
	{departure date (datetime)} = {depart (datetime)}	Y
	{passenger (integer)} = {adult (integer), child (integer), infant (integer)}	P
	{from (string), to (string)} = {departure city (string), arrival city (string)}	Y
	{from (string)} = {depart (string)}	Y
	{return date (datetime)} = {return (datetime)}	Y
Movies	{artist (any)} = {actor (any)} = {star (any)}	Y
	{genre (string)} = {category (string)}	Y
	{cast & crew (any)} = {actor (any), director (any)}	Y

Figure 3.9: Experimental results for Airfares and Movies.

occurrences in different schemas). Therefore, *target precision* and *target recall* of $\mathcal{M}_h$ with respect to $\mathcal{M}_c$ are:

$$P_T(\mathcal{M}_h, \mathcal{M}_c) = \sum_{A_j} \frac{O_j}{\sum O_k} P_{A_j}(\mathcal{M}_h, \mathcal{M}_c)$$

$$R_T(\mathcal{M}_h, \mathcal{M}_c) = \sum_{A_j} \frac{O_j}{\sum O_k} R_{A_j}(\mathcal{M}_h, \mathcal{M}_c).$$

3.5.2 Experimental Results

To illustrate the effectiveness of the mining approach, we only list and count the "semantically difficult" matchings discovered by the correlation mining algorithm, not the simple matchings by the syntactic

Domain	P_T (20%)	R_T (20%)	P_T (10%)	R_T (10%)
Books	1	1	1	1
Airfares	1	1	1	0.71
Movies	1	1	1	1
MusicRecords	1	1	0.76	1
Hotels	0.86	1	0.86	0.87
CarRentals	0.72	1	0.72	0.60
Jobs	1	0.86	0.78	0.87
Automobiles	1	1	0.93	1

Figure 3.10: Target accuracy of 8 domains.

merging in the data preparation (*e.g.*, {title of book} to {title}). Our experiment shows that many frequently observed matchings are in fact "semantically difficult" and thus cannot be found by syntactic merging.

Result on the TEL-8 Dataset: In this experiment, we run our algorithm (with H-measure as the correlation measure) on the TEL-8 dataset. We set the thresholds T_p to 0.85 and T_d to 0.6 for positive correlation mining and T_n to 0.75 for negative correlation mining. We empirically get these numbers by testing the algorithm with various thresholds and choose the best one. As Section 8 will discuss, a more systematic study can investigate in choosing appropriate threshold values.

Figure 3.8 illustrates the detailed results of n-ary complex matchings discovered in the Books domain. The step of group discovery found 5 likely groups (G_1 to G_5 in Figure 3.8). In particular, m_p gives a high value (actually the highest value) for the group of last name (any) and first name (any). The matching discovery found 6 likely complex matchings (M_1 to M_6 in Figure 3.8). We can see that M_1 and M_3 are fully correct matchings, while others are partially correct or incorrect. Last, the matching selection will choose M_1 and M_3 (*i.e.*, the correct ones) as the final output.

Figure 3.9 shows the results on Airfares and Movies. (The results of other domains can be found in the extended report [38]). The third column denotes the correctness of the matching. In particular, Y means a fully correct matching, P a partially correct one and N an incorrect one. Our mining algorithm

does find interesting matchings in almost every domain. For instance, in the Airfares domain, we find 5 fully correct matchings, *e.g.*, {destination (string)} = {to (string)} = {arrival city (string)}. Also, {passenger (integer)} = {adult (integer), child (integer), infant (integer)} is partially correct because it misses senior (integer).

Since, as a statistical method, our approach relies on "sufficient observations" of attribute occurrences, it is likely to perform more favorably for frequent attributes (*i.e.*, the head-ranked attributes in Figure 3.5). To quantify this "observation" factor, we report the target accuracy with respect to the attribute frequencies. In particular, we consider the attributes above a *frequency threshold* T (*i.e.*, the number of occurrences of the attribute over the total number of schemas is above T) in both discovered matchings and correct matchings to measure the target accuracy. Specifically, we run the algorithms on all the attributes and then report the target accuracy in terms of the frequency-divided attributes. In the experiment, we choose T as 20% and 10%.

Consider the Airfares domain, if we only consider the attributes above 20% frequency in the matching result, only 12 attributes are above that threshold. The discovered matchings in Figure 3.9 become {destination (string)} = {to (string)}, {departure date (datetime)} = {depart (datetime)}, and {return date (datetime) = return (datetime)}. (The other attributes are removed since they are all below 20% frequency.) These three matchings are exactly the correct matchings the human expert can recognize among the 12 attributes and thus we get 1.0 in both target precision and recall.

Next, we apply the 10% frequency threshold and get 22 attributes. The discovered matchings in Figure 3.9 are unchanged since all the attributes (in the matchings) are now passing the threshold. Compared with the correct matchings among the 22 attributes, we do miss some matchings such as {cabin (string)} = {class (string)} and {departure (datetime) = departure date (datetime)}. Also, some

Domain	reduced false positive (20%)	missed false positive (20%)	reduced false positive (10%)	missed false positive (10%)
Books	0	0	3	0
Airfares	2	0	22	0
Movies	0	0	2	0
MusicRecords	3	0	5	1
Hotels	6	1	11	2
CarRentals	2	1	2	1
Jobs	4	0	9	1
Automobiles	0	0	2	1

Figure 3.11: The effectiveness of reducing false matchings in the matching selection step.

matchings are partially correct such as {passenger (integer)} = {adult (integer), child (integer), infant (integer)}. Hence, we get 1.0 in target precision and 0.71 in target recall.

Figure 3.10 lists the target accuracies of the 8 domains under thresholds 20% and 10%. From the result, we can see that our approach does perform better for frequent attributes.

Evaluating the Matching Selection Strategy: To evaluate the effectiveness of the matching selection algorithm we developed in Section 3.2.2, which exploits conflict between matching candidates to remove false positives, we count the number of false matchings reduced and missed by the selection step respectively. Figure 3.11 lists this result for the eight domains under both 20% and 10% frequency thresholds. We can see that the greedy selection strategy based on C_{max} measure is quite effective in reducing false matchings. Most false matchings are removed in the selection step. In particular, although the 10% frequency threshold may result in more false matchings comparing to the 20% one, the selection strategy can remove most of them and keep the performance good. For instance, in the Airfares domain under 10% frequency threshold, 22 false matchings are removed and no false matching is missed.

Evaluating the Data Preprocessing Step: To evaluate the effectiveness of the data preprocessing step, we test the DCM algorithm over schemas without data preprocessing. In particular, we only perform the

Domain	P_T (20%)	R_T (20%)	P_T (10%)	R_T (10%)
Books	*0.79* (**-0.21**)	1	*0.74* (**-0.26**)	1
Airfares	1	1	*0.81* (**-0.19**)	*0.82* (+0.11)
Movies	1	1	*0.87* (**-0.13**)	1
MusicRecords	*0.93* (**-0.07**)	1	*0.70* (**-0.06**)	1
Hotels	*0.66* (**-0.20**)	1	*0.47* (**-0.39**)	*0.46* (**-0.41**)
CarRentals	*1* (+0.28)	*0.63* (**-0.37**)	*1* (+0.28)	*0.16* (**-0.44**)
Jobs	*0.70* (**-0.30**)	*1* (+0.14)	*0.52* (**-0.26**)	0.87
Automobiles	1	1	*0.66* (**-0.27**)	*0.68* (**-0.32**)

Figure 3.12: Target accuracy of the 8 domains without data preprocessing.

standard normalization sub-step in Section 3.4 for the input schemas and ignore the type recognition and syntactic merging sub-steps. Our goal is to see the impact of these sub-steps on the accuracy of matching.

Intuitively, although query interfaces are quite concerted in terms of the attributes they use, there still are many syntactic variations for expressing the same attribute, *e.g.*, title, book title, title of book and search by title for attribute "title." As discussed in Section 3.4, type recognition and syntactic merging can help merge these variations into a single attribute and thus increase attribute occurrences across query interfaces, which can improve the performance of the subsequent correlation mining algorithm.

Figure 3.12 shows the result of running the DCM algorithm with non-preprocessed schemas as input. In Figure 3.12, we write accuracies that change after removing the data preprocessing step in italic font and show the differences in brackets. As we can see, the accuracies for many domains are much worse than the ones with data preprocessing in Figure 3.10. In particular, under the 10% frequency threshold, where more attributes are considered for mining matchings, accuracies are greatly reduced. Therefore, applying the data preprocessing step, although may itself result in some errors, is crucial for our mining-based matching approach and can indeed significantly enhance the matching accuracy.

Domain	$P_T(H)$ (10%)	$R_T(H)$ (10%)	$P_T(\zeta)$ (10%)	$R_T(\zeta)$ (10%)
Books	1	1	*0.80* (**-0.20**)	1
Airfares	1	0.71	*0.79* (**-0.21**)	*0.61* (**-0.10**)
Movies	1	1	*0.93* (**-0.07**)	1
MusicRecords	0.76	1	0.76	1
Hotels	0.86	0.87	*0.44* (**-0.42**)	*0.95* (+0.08)
CarRentals	0.72	0.60	*0.68* (**-0.04**)	*0.62* (+0.02)
Jobs	0.78	0.87	*0.64* (**-0.14**)	0.87
Automobiles	0.93	1	*0.78* (**-0.15**)	1

Figure 3.13: Comparison of H-measure and *Jaccard*.

Evaluating the H-Measure: We compare the H-measure with other measures and the result shows that H-measure gets better target accuracy. As an example, we choose *Jaccard* (ζ) as the measure we compare to. With *Jaccard*, we define the m_p and m_n as

$$m_p(A_p,A_q) = \begin{cases} \zeta(A_p,A_q), & \frac{f_{11}}{f_{++}} < T_d \\ 0, & otherwise, \end{cases}$$

and

$$m_n(A_p,A_q) \quad = \quad 1-\zeta(A_p,A_q).$$

We set the T_p and T_n for this *Jaccard*-based m_p and m_n as 0.5 and 0.9 respectively. We compare the target accuracy of H-measure and *Jaccard* in the situation of 10% frequency threshold. The result (Figure 3.13) shows that H-measure is better in both target precision and target recall in most cases. Similar comparisons show that H-measure is also better than other measures such as *Cosine* and *Confidence*.

3.6 Conclusion

This chapter explores co-occurrence patterns among attributes to tackle the complex matching problem. Specifically, we abstract complex matching as correlation mining and develop the DCM framework. Further, we propose a new correlation measure, H-measure, for mining negative correlations. Our experiments validate the effectiveness of both the mining approach and the H-measure.

To complete an automatic matching process, which starts from raw HTML pages, we integrate the DCM framework with an automatic interface extractor [72]. Such "system integration" turns out to be non-trivial– As automatic interface extraction cannot be perfect, it will introduce "noise" (*i.e.*, erroneous extraction), which challenges the performance of the subsequent matching algorithm. Chapter 4 will discuss our approach to maintaining the matching quality with the presence of errors in the interface extraction step.

Chapter 4

Dealing with Noise: the Ensemble DCM Framework

Our study on holistic matching algorithms so far has been focused on the matching task in isolation– That is, we assume the input schemas are perfectly extracted. To complete an automatic matching process, we must incorporate automatic techniques for interface extraction. Executing the DCM framework on automatically extracted interfaces, we find that the inevitable errors in automatic interface extraction may significantly affect the matching result. To make the DCM framework robust against such "noisy" schemas, we propose to integrate it with an "ensemble" approach, which creates an ensemble of DCM matchers, by randomizing the schema data into many *trials* and aggregating their ranked results by taking majority voting.

4.1 The Ensemble DCM Framework

To fully automate the DCM matching process, which starts from raw HTML pages as Figure 3.1 shows, we must integrate the DCM framework (discussed in Chapter 3) with an automatic interface extractor.

It turns out that such integration is not trivial– As automatic interface extractor *cannot* be perfect, it will introduce "noise," which challenges the performance of the subsequent DCM matching algorithm. This chapter presents a refined algorithm, the *ensemble* DCM framework, in contrast to the *base* DCM framework in Section 3, to *maintain* the robustness of DCM against such noisy input.

We note that such "system integration" issues have not been investigated in earlier works. Most works on matching query interfaces, for instance our MGS and DCM frameworks and others [41, 66], all adopt manually extracted schema data for experiments. While these works rightfully focus on isolated study of the matching module to gain specific insight, for our goal of constructing a fully automatic matching process, we must now address the robustness problem in integrating the interface extraction step and the matching algorithm.

In particular, we integrate our DCM algorithm with the interface extractor we developed recently [72], which tackles the problem of interface extraction with a parsing paradigm. The interface extractor as reported in [72] can typically achieve 85-90% accuracy– thus it will make about 1-1.5 mistake for every 10 query conditions to extract. While the result is quite promising, the 10-15% errors (or *noise*) may still affect the matching quality. As our experiment shows in Section 4.5, with noisy schemas as input, the accuracy of the base DCM framework may degrade up to 30%.

The performance degradation results mainly from two aspects: First, noise may affect the qualification of some correlations and decrease their C_{min} values (*i.e.*, Equation 3.1) below the given threshold. In this case, the dual correlation mining algorithm cannot discover those matchings. Second, noise may affect the right ranking of matchings (with respect to the C_{max} measure, *i.e.*, Equation 3.2) and consequently the result of greedy matching selection. Although in principle both qualification and ranking can be affected, the influence on qualification is not as significant as on ranking. Matching qualification will be affected when there are enough noisy schemas, which make the C_{min} value lower than the given

thresholds T_p or T_n. In many cases when only a little noise exists, the affected matchings are still above the threshold and thus can be discovered in the qualification step. However, the ranking of matchings using C_{max} is more subtle– That is, even when there are only little noise, the ranking of matchings is still likely to be affected (*i.e.*, incorrect matchings maybe ranked higher than correct ones). The reason is that other than comparing matchings to a fixed threshold, the ranking step needs to compare matching among themselves. A single noise is often enough to change the ranking of two conflict matchings. Consequently, the ranking is less reliable for the matching selection step to choose correct matchings. As a result, although correct matchings may be discovered by the dual correlation mining process, they may be pruned out by the matching selection phase due to the incorrect ranking of matchings, and thus the final matching accuracy degrades.

While large scale schema matching brings forward the inherent problem of noisy quality in interface extraction, the large scale also lends itself to an intriguing potential solution. An interesting question to ask is: *Do we need all input schemas in matching their attributes*? In principle, since pursuing a correlation mining approach, our matching techniques exploit "statistics-based" evaluation in nature and thus need only "sufficient observations." As query interfaces tend to share attributes, *e.g.*, author, title, subject, ISBN are repeatedly used in many book sources, a subset of schemas may still contain sufficient information to "represent" the complete set of schemas. Thus, the DCM matcher in fact needs only sufficient correct schemas to execute, instead of all of them. This insight is promising, but it also brings a new challenge: As there is no way to differentiate noisy schemas from correct ones, how should we select input schemas to guarantee the robustness of our solution?

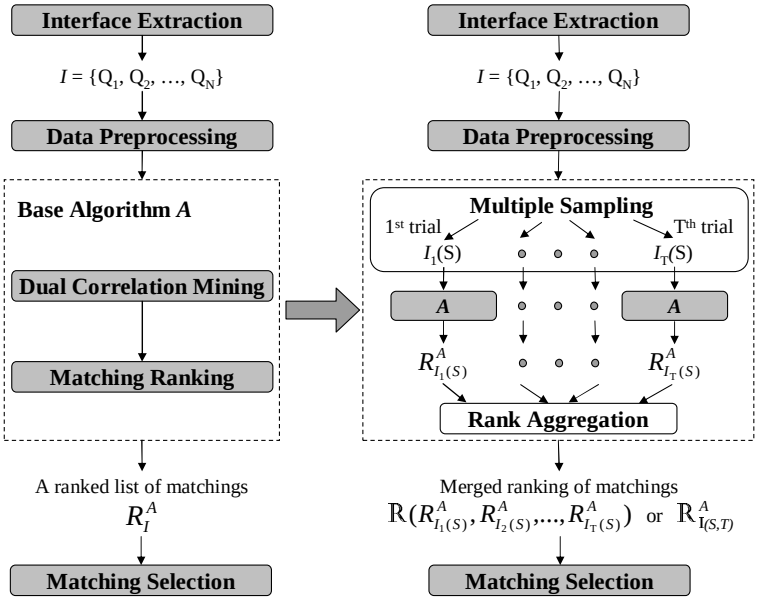
Tackling this challenge, we propose to extend DCM in an *ensemble* scheme with sampling and voting techniques. (Figure 4.1 shows this extension from base DCM framework, *i.e.*, Figure 4.1(a), to ensemble DCM framework, *i.e.*, Figure 4.1(b), which we will elaborate in Section 4.2.) To begin with, we consider

to execute the DCM matcher on a randomly sampled subset of input schemas. Such a *downsampling* has two attractive characteristics: First, when schemas are abundant, the downsampling is likely to still contain sufficient correct schemas to be matched. Second, by sampling away some schemas, it is likely to contain less noise and thus is more probable to sustain the DCM matcher. (Our analysis in Section 4.2 attempts to build analytic understanding of these "likelihoods.")

Further, since a single downsampling may (or may not) achieve good result, as a randomized scheme, its expected robustness can only be realized in a "statistical" sense– Thus, we propose to take an ensemble of DCM matchers, where each matcher is executed over an independent downsampling of schemas. We expect that the majority of those ensemble matchers on randomized subsets of schemas will perform more reliably than a single matcher on the entire set. Thus, by taking majority voting among these matchers, we can achieve a robust matching accuracy.

We note that, our ensemble idea is inspired by *bagging classifiers* [11, 26] in machine learning. Bagging is a method for maintaining the robustness of "unstable" classification algorithms where small changes in the training set result in large changes in prediction. In particular, it creates multiple versions of a classifier, trains each classifier on a random redistribution of the training set and finally takes a plurality voting among all the classifiers to predict the class. Therefore, our ensemble approach has the same foundation as bagging classifiers on exploiting majority voting to make an algorithm robust against outlier data in the input.

However, our approach is different from bagging classifiers in several aspects. First, *setting*: We apply the idea of the ensemble of randomized data for unsupervised learning (*e.g.*, in our scenario, schema matching with statistical analysis), instead of supervised learning (*i.e.*, human experts give the learner direct feedback about the correctness of the performance [45]), which bagging classifiers is developed for. Second, *techniques*: Our concrete techniques are different from bagging classifiers. In particular,



(a) The Base **DCM** Framework (b) The Ensemble **DCM** Framework

Figure 4.1: From the base DCM framework to the ensemble DCM framework.

in the sampling part, we take a downsampling other than random redistribution with replacement; in the voting part, we need to aggregate a set of ranked lists, which is more complicated than aggregate a set of labels in bagging classifiers. Third, *analytic modeling*: We build an analytic modeling specific to our matching scenario (Section 4.2), which enables us to validate the effectiveness of a particular configuration and thus can be the basis for the design of the ensemble scheme.

We will next discuss this ensemble DCM framework in detail. In particular, we first more formally model this framework and analyze its effectiveness (Section 4.2). Then, we aggregate the results of multiple DCM matchers with a voting algorithm, which thus essentially captures the consensus of the majority (Section 4.4).

4.2 Analytical Modeling

We develop a general modeling to formalize the ensemble DCM framework just motivated. Our goals are two fold: First, based on our modeling, we can analytically judge the effectiveness of the ensemble

approach. Second, the modeling can be used to validate the setting of parameters in the ensemble scheme.

We first redraw the base DCM framework in Figure 3.1 as Figure 4.1(a) by expanding the two steps in matching construction, *i.e.*, matching ranking and matching selection. We view the dual correlation mining algorithm N-ARYSCHEMAMATCHING and the matching ranking together as a black box *base algorithm A*. As we just discussed, the performance degradation is mainly caused by the impact of noise on *A*, where the output of *A*, denoted by R_I^A (*i.e.*, the output ranking determined by *A* over input *I*), is disturbed. The goal of our ensemble DCM framework is thus to make *A* still output reasonably good ranking of matchings with the presence of noise.

Specifically, given a set of N schemas I as input, assume there are W problematic schemas (*i.e.*, noise) that affect the ranking of M. Suppose the holistic matcher A can correctly rank M if one trial draws no more than K noise ($K < W$)– *i.e.*, in which case, M as a correct matching can actually be ranked higher.

Next, we need to model the ensemble framework, which consists of two steps: *multiple sampling* and *rank aggregation*, as Figure 4.1(b) shows. *First*, in the multiple sampling step, we conduct T downsamplings of the input schemas I, where each downsampling is a subset of independently sampled S schemas from I. We name such a downsampling as a *trial* and thus have T trials in total. We denote ith trial as $I_i(S)$ $(1 \leq i \leq T)$. By executing the base algorithm A over each trial $I_i(S)$, we get a ranked list of matchings $R^A_{I_i(S)}$. *Second*, the rank aggregation step aggregates ranked matchings from all the trials, *i.e.*, $R^A_{I_i(S)}$ $(1 \leq i \leq T)$, into a merged list of ranked matchings, which we denote as $\mathbb{R}(R^A_{I_1(S)}, ..., R^A_{I_T(S)})$, or $\mathbb{R}^A_{\mathbb{I}(S,T)}$ in short. We expect the aggregate ranking $\mathbb{R}^A_{\mathbb{I}(S,T)}$ can alleviate the impact of noise and thus is better than R^A_I.

Since W is determined by "inherent" characteristics of input schemas I and K by the holistic matcher A, we name them as *base parameters*. Unlike W and K, the sampling size S and the number of trials T are "engineered" configurations of the ensemble framework and thus named as *framework parameters*.

Our goal of analysis is thus to justify, given estimation of the base parameters, W and K, which characterize the data quality and the base algorithm, can certain configuration, in terms of S and T, of the ensemble framework achieve robustness? (If so, we will then ask, how to determine appropriate settings of S and T, which is the topic of Section 4.3.)

In particular, given our modeling, we can derive the probability to correctly rank M in a single trial, which we name as *hit probability*, *i.e.*, the chance of "hit" a correct ranking of M in a single trial (and as we will discuss later, we will do more trials to enhance the overall hit ratio). Given base parameters W and K of M, hit probability is a function of S (and not T as it is for a single trial) and thus denoted as $\alpha_M(S)$. To derive $\alpha_M(S)$, we first compute the probability that there are exactly i noise in a single trial, denoted by $Pr(k=i|S)$, *i.e.*, with i noise out of W and $S-i$ correct ones out of $N-W$:

$$Pr(k=i|S) = \frac{\binom{W}{i}\binom{N-W}{S-i}}{\binom{N}{S}} \tag{4.1}$$

As our model assumes, M can be correctly ranked when there are no more than K noise. We thus have:

$$\alpha_M(S) = \sum_{i=0}^{K} Pr(k=i|S) \tag{4.2}$$

Next, we are interested in how many times, among T trials, can we observe M being ranked correctly? (This derivation will help us to address the "reverse" question in Section 4.3: To observe M in a majority of trials with a high confidence, how many trials are necessary?) This problem can be transformed as the standard scenario of tossing an unfair coin in statistics: Given the probability of getting a "head" in each toss as $\alpha_M(S)$, with T tosses, how many times can we observe heads? With this equivalent view, we know that the number of trials in which M is correctly ranked (*i.e.*, the number of tosses to observe heads), denoted by O_M, is a random variable that has a binomial distribution [3] with the success probability in one trial as $\alpha_M(S)$. We use $Pr(O_M = t|S,T)$ to denote the probability that M is correctly ranked in exactly t trials. According to the binomial distribution, we have

$$Pr(O_M = t|S,T) = \frac{T!}{t!(T-t)!}\alpha_M(S)^t(1-\alpha_M(S))^{T-t} \tag{4.3}$$

Since our goal is to take majority voting among all the trials (in rank aggregation), we need a sufficient number of trials to ensure that M is "very likely" to be correctly ranked in the relative majority of trials. As an analogy, consider the coin tossing: Even the probability to get a head in each toss is high, say 0.8, we may not always observe $0.8 \times T$ heads in T trials; the actual number of heads may even be a minority of trials– And our goal is to design a T such that "the number of heads" is very likely to be the majority. We thus need a sufficient number of trials to enable the majority voting. We name the probability that M can be correctly ranked in the majority of trials (*i.e.*, more than half of trials) as *voting confidence*. Voting confidence is a function of T (as just intuitively observed) and S (as it also depends on $\alpha_M(S)$ and thus S). We denote the voting confidence as $\beta_M(S,T)$. In particular, we have

$$\beta_M(S,T) = \sum_{t=\frac{T+1}{2}}^{T} Pr(O_M = t|S,T). \tag{4.4}$$

As a remark, in Equation 4.4, we constrain T as an odd number and thus $\frac{T+1}{2}$ is the minimum number of trials needed to be the majority[1].

Our modeling essentially captures the functional relationship of the sampling size S and the number of trials T to together achieve a desired voting confidence. There are two interpretations of Equation 4.4 in examining a framework: First, given S and T, we can use Equation 4.4 to evaluate how effective the framework is. In particular, we illustrate with Example 19 as a basis of understanding how the framework works. Second, we can use Equation 4.4 to design the configuration of a framework. That is, for an objective voting confidence to achieve, what would be the right configuration of S and T? Section 4.3 will focus on this configuration issue.

Example 19: Assume there are 50 input schemas (*i.e.*, $N = 50$). As characteristics of the data quality and the base algorithm, suppose a matching M cannot be correctly ranked because of 6 noisy schemas (*i.e.*, $W = 6$); on the other hand, suppose M can be correctly ranked if there are no more than two noisy schemas (*i.e.*, K = 2). Also, as the configuration of the ensemble framework, suppose we want to sample 20 schemas in a single trial and conduct 99 trials (*i.e.*, $S = 20$ and $T = 99$).

According to Equation 4.1, in any single trial, we have 0.04 probability to get no noisy schema, 0.18 probability with one and 0.33 probability with two. Together, we have 0.04 + 0.18 + 0.33 = 0.55 probability to correctly rank M in one trial (*i.e.*, $\alpha_M(S) = 0.55$).

Further, Figure 4.2 shows the binomial distribution of O_M. Going back to the coin tossing analogy, this figure essentially shows, if the probability to get a head in one toss is 0.55, after tossing 99 times, the probability of observing a certain number of heads. For instance, we have $Pr(O_M = 50|S,T) = 0.05$, which means the probability to observe 50 heads in 99 tosses is 0.05. According to Equation 4.4, we

[1]When T is odd, the notion of majority is always well defined, as there are no ties (of equal halves). This advantage ensures there is no ambiguous situation in comparing two matchings in the rank aggregation step in Section 4.4. Also, when T is odd, $\beta_M(S,T)$ becomes a monotonic function of T. We use this property to derive an appropriate configuration in Section 4.3.

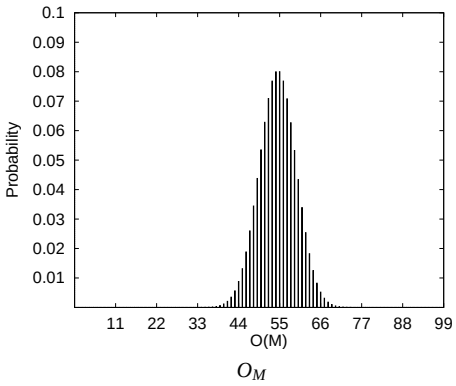


Figure 4.2: The binomial distribution of O_M, with $T = 99$ and $\alpha_M(S) = 0.55$.

have 0.84 voting confidence to correctly rank M (or observe heads) in more than 49 trials (or tosses) (*i.e.*, $\beta_M(S,T) = 0.84$). Therefore, even $\alpha_M(S)$ is not very high, *e.g.*, 0.55 in this example, with sufficient number of trails, it is still very likely that M can be correctly ranked in the majority of trials.

Finally, while our analysis above focuses on a single matching, there are multiple matchings, M_1, M_2, ..., M_n, to discover. We note that our analysis can generally assume a representative "worst-case" matching, based on which the analysis will also cover all the matchings. Specifically, the above modeling can be applied to any M_i with its corresponding W_i and K_i values. We then assume there is a "worst-case" matching M^* with base parameters W^* and K^*. We want to show that if we are likely to correctly rank M^* in the majority of trials under some setting, we are even more likely to correctly rank all the matchings M_1, M_2, ..., M_n in the majority of trials with the same setting. If this statement can be justified, we only need to consider the "worst-case" situation in determining the ensemble configuration in Section 4.3.

We show that the base parameters of the imaginary "worst-case" matching M^* can be set as $W^* = \max W_i$ and $K^* = \min K_i$, $1 \leq i \leq n$. Intuitively, the higher W is, the lower $\alpha_M(S)$ will be because we have more noise in the input schemas I; on the other hand, the lower K is, the lower $\alpha_M(S)$ will be because the base algorithm A is less robust against noise. More formally, we can show that $\alpha_M(S)$

is monotonically decreasing with respect to W and monotonically increasing with respect to K. (The derivation is straightforward and thus we do not provide a proof here.) Therefore, if we assume a matching M^* with base parameters W^* as the maximal value of W_i and K^* the minimal value of K_i, we have $\alpha_{M_i}(S) \geq \alpha_{M^*}(S)$ any matching M_i ($1 \leq i \leq n$).

Further, we can show that all the matchings also have higher voting confidence than M^*. Intuitively, if a matching M has higher hit probability, M should be more likely to be observed in the majority of trials, which means it also has a higher voting confidence. In particular, we can show that $\beta_M(S,T)$ is monotonically increasing with respect to $\alpha_M(S)$. (Similarly, the derivation is straightforward and thus we do not provide a proof here.) Therefore, since $\alpha_{M_i}(S) \geq \alpha_{M^*}(S)$ ($1 \leq i \leq n$), we have $\beta_{M_i}(S,T) \geq \beta_{M^*}(S,T)$ ($1 \leq i \leq n$). This inequality indicates that M^* is indeed the "worst-case" matching. Specifically, if we can find an appropriate setting of S and T to correctly rank M^* in the majority of trials with high confidence, we will have even more confidence to correctly rank all the matchings in the majority of trials with the same setting of S and T.

4.3 Sampling and Trials: Configuration

This section focuses on the first phase of the ensemble framework: Sampling and trials. The key challenge we need to address is: Given W and K, we need to find an appropriate configuration of S and T to provide guarantee on voting confidence. To begin with, we must characterize our "system environment" by estimating the base parameters W and K. Then, we discuss our strategy to configure S and T based on our modeling in Section 4.2.

Base Parameters: Before deriving S and T, we need to estimate the "worst-case" base parameters W^* and K^* in Equations 4.1 and 4.2. In particular, W^* and K^* can be related to the error rate and the tolerance threshold respectively in the modeling of error cascade. First, as W^* characterizes the noisy

degree of the input schemas I, we let $W^* = N \times \rho$, where N is the number of schemas and ρ is the error rate of I. In our development, we set ρ to 0.1, as the worst-case value, according to the accuracy of current interface extraction technique, as discussed earlier. Second, since the behavior of A is very specific and complicated, it may be difficult to accurately obtain K^*. We thus take a conservative setting, which will lead to a "safe" framework, *e.g.*, setting the worst-case K^* as a small constant.

As just discussed, all matchings that are not worse than the worst-case setting can be guaranteed to have higher voting confidences. Therefore, with conservative worst-case settings, we expect to correctly rank more matchings in the aggregate result $\mathbb{R}^A_{\mathbb{I}(S,T)}$.

Framework Parameters: In Section 4.2, we have shown that, for some matching M with respect to given base parameters W and K, for certain framework parameters S and T, we can derive the voting confidence $\beta_{M^*}(S,T)$ with statistical analysis. Now we are facing the reverse problem: Given estimated W, K, and our objective voting confidence, what are the appropriate S and T values we should take? Formally, *given W, K, and an objective voting confidence c, what are the sampling size S and the number of trials T we should take to ensure M^* has at least a probability of c to be correctly ranked in the majority of trials, i.e.,* $\beta_{M^*}(S,T) \geq c$?

In particular, we want to know, among all the (S,T) pairs that satisfy the above statement, which pair is the most appropriate? To answer this question, we need to develop some criteria to evaluate settings. Intuitively, we would like to prefer a (S,T) pair that can maximize S and minimize T:

On the one hand, we want to reduce unnecessary downsampling. A very small S value may not be able to collect enough schemas to represent the complete input data and consequently degrade the accuracy of the base algorithm A. It may also, by overly-aggressive downsampling, remove some more "unique" (but correct) schemas from consideration, and thus reduce the applicability of the matching result. Thus, among all the valid (S,T) pairs, we prefer a larger S that can cover more schemas.

On the other hand, we want to reduce unnecessary trials. As Section 4.2 discussed, the more trials we have, the higher voting confidence will be. We can formally show that when T is limited to be odd, $\beta_{M^*}(S,T)$ is monotonically increasing with respect to T. (Again, the derivation is straightforward and thus we do not provide a proof here.) Considering the execution time of the ensemble framework, we do not want to be over-tried; therefore, among all the valid (S,T) pairs, we prefer a pair with a smaller T.

However, these two goals cannot be optimized at the same time, because as our modeling shows, S and T are not independent– One will negatively affect the choice of another. Specifically, when we set $\beta_{M^*}(S,T)$ to an objective confidence c, T can be viewed as a function of S or vice versa. Choosing one will thus also affect another: A larger S will result in a lower hit probability and thus more trials T for the same objective confidence; on the other hand, a smaller T will demand a higher hit probability and thus a smaller sampling size S. Consequently, in the space of all valid (S,T) pairs, there does not exist one that can optimize both S and T.

To balance these two goals, we have to choose a trade-off setting. We propose two ways to determine S and T:

First, $S \rightarrow T$: In some situations, we may have a reasonable grasp of S, so that we know the range of input size (*i.e.*, the degree of downsampling) that the base algorithm may demand– *e.g.*, some statistical approach typically requires a minimal number of observations of data to ensure its statistical confidence. In such a case, we can start with an S value and set T as the minimal (odd) number that can achieve the objective confidence c, *i.e.*,

$$T = \min\{t | t > 0, t \text{ is odd}, \beta_{M^*}(S,T)(S,t) \geq c\} \tag{4.5}$$

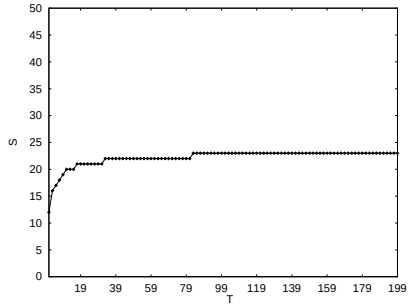


Figure 4.3: The insensitivity of S on T.

Second, $T \rightarrow S$: In other situations, we may be constrained by affordable computation time, which determines the acceptable range of T. In this case, we start with a desired number of trials T and choose the maximal S to achieve the objective confidence, *i.e.*,

$$S = \max\{s | 1 \leq s \leq N, \beta_{M^*}(S,T)(s,T) \geq c\} \tag{4.6}$$

Example 20: Assume there are 50 input schemas (*i.e.*, $N = 50$) and our objective confidence is 0.9 (*i.e.*, $c = 0.9$). According to our discussion, the settings of the "worst-case" matching M^* are $W^* = N \times \rho = 50 \times 0.1 = 5$ and $K^* = 2$. Setting K^* to 2 is a "safe" configuration we also use in our experiments (Section 3.5).

In the $S \rightarrow T$ strategy, assume we set S = 20. Based on our modeling, for any odd number t, we can compute the voting confidence $\beta_{M^*}(S,t)$. According to Equation 4.5, we take the minimal t that satisfies $\beta_{M^*}(S,t) \geq 0.9$ and thus we get $T = 11$.

On the other hand, in the $T \rightarrow S$ strategy, assume we set T = 41. Similarly, for any s ($1 \leq s \leq N$), we can compute $\beta_{M^*}(s,T)$. According to Equation 4.6, we take the maximal S that satisfies $\beta_{M^*}(s,T) \geq 0.9$ and thus we get $S = 22$.

Although both $S \rightarrow T$ and $T \rightarrow S$ are valid configuration strategies, as Example 20 just showed, in practice the $T \rightarrow S$ strategy is better because it is easier to pick T. To illustrate this statement, we

compute the corresponding S values for all the odd numbers T between 0 to 200 using the $T \rightarrow S$ strategy, *i.e.*, Equation 4.6, with the same system setting as Example 20 assumed. Figure 4.3 shows the result, from where we observe that when T increase to some point, around 30 in this example, the corresponding S values become very stable, almost insensitive to the change of T.

On the other hand, from the same Figure 4.3, we can infer the opposite trend of the $S \rightarrow T$ strategy. Picking an S will significantly affect the value of T. Some S values may result in a very large T, which is not affordable in practice. In some cases, for a large S, maybe it is even impossible to find a corresponding T that satisfies the given objective voting confidence.

Overall, it is much easier to pick T than S in practice. Therefore, in our experiments (Section 3.5), we adopt the $T \rightarrow S$ strategy. Also, we will show that the empirical result of testing the framework with various configuration settings is consistent with our analysis above.

4.4 Voting: Rank Aggregation

This section discusses the second phase of the ensemble framework: Aggregating rankings $R^A_{I_1(S)}$, ..., $R^A_{I_T(S)}$ from the T trials into a merged list of ranked matchings $\mathbb{R}^A_{\mathbb{I}(S,T)}$. The main issue we are facing is to develop a rank aggregation strategy that can reflect the majority "consensus" in $\mathbb{R}^A_{\mathbb{I}(S,T)}$.

We notice that this rank aggregation in our situation is slightly different from the traditional rank aggregation problem. Traditional rank aggregation assumes all voters share the same set of candidates and only rank them in different orders. In contrast, in our scenario, no candidate is given before executing the base algorithm and each trial outputs its own matching result. Therefore, before aggregate rankings, we need to have a candidate selection step to select matching candidates.

Consequently, the rank aggregation phase consists of two sub-steps: 1) Candidate selection: To select candidates from each $R^A_{I_i(S)}$ to form a common pool of candidates $\mathcal{C}$. 2) Rank aggregation: To

aggregate the T rankings $PR^A_{I_1(S)}$, ..., $PR^A_{I_T(S)}$ into $\mathbb{R}^A_{\mathbb{I}(S,T)}$, where $PR^A_{I_i(S)}$ is the "projected" ranking of $R^A_{I_i(S)}$ on $\mathcal{C}$, as we will discuss below.

Candidate Selection

We select candidates based on the intuition that if a matching M is only discovered by a minority of trials, M is more likely to be a false matching. Therefore, we consider a matching as a candidate if it appears in the majority of T rankings, $R^A_{I_1(S)}$, ..., $R^A_{I_T(S)}$. All the matchings whose number of occurrences are less than $\frac{T+1}{2}$ are thus pruned.

Let $\mathcal{C}$ denote the union of all the candidates in each $R^A_{I_i(S)}$. After candidate selection, we will remove the non-candidate matchings from each $R^A_{I_i(S)}$ and meanwhile preserving the ordering of candidates; the corresponding new ranked list, which can be viewed as a "projection" of $R^A_{I_i(S)}$ on $\mathcal{C}$, contains only candidates and is denoted as $PR^A_{I_i(S)}$.

Example 21: Assume we execute the base algorithm A on three trials, *i.e.*, $T = 3$, and the outputs are thus three ranked lists $R^A_{I_1(S)}$, $R^A_{I_2(S)}$ and $R^A_{I_3(S)}$. Suppose $R^A_{I_1(S)}$ outputs ranking $M_1 > M_2 > M_3 > M_4$ in descending order, $R^A_{I_2(S)}$ outputs $M_2 > M_1 > M_3 > M_5$, and $R^A_{I_3(S)}$ outputs $M_3 > M_1 > M_2 > M_4$.

Since $\frac{T+1}{2} = 2$, any matching that occurs only once will be pruned. In particular, M_5 is pruned; other matchings, M_1, M_2, M_3 and M_4, all at least occur twice and thus are selected as matching candidates. Therefore, we have $\mathcal{C} = \{M_1, M_2, M_3, M_4\}$.

The projected rankings are thus $RR^A_{I_1(S)}$: $M_1 > M_2 > M_3 > M_4$, $PR^A_{I_2(S)}$: $M_2 > M_1 > M_3$, and $PR^A_{I_3(S)}$: $M_3 > M_1 > M_2 > M_4$. In particular, M_5 does not appear in $PR^A_{I_2(S)}$ because it has been pruned.

Rank Aggregation

In rank aggregation, we need to construct an ordered list $\mathbb{R}^A_{\mathbb{I}(S,T)}$ for the candidates in $\mathcal{C}$, based on the individual ranks $PR^A_{I_1(S)}$, ..., $PR^A_{I_T(S)}$. This problem is essentially a *rank aggregation* problem, which

has been extensively studied as a particular *voting* system in both social science [44, 69] and computer science [29, 31]. In the literature, many rank aggregation strategies have been proposed, such as Borda's aggregation [10], Kemeny optimal aggregation [44], and footrule optimal aggregation [29]. There does not exist an aggregation strategy that can beat other strategies in all aspects– Different strategies have different strength and weakness.

Before discussing concrete aggregation strategies, we first need to solve the partial list problem. Specifically, since the output of one trial may not contain all the candidates in C, $PR^A_{I_i(S)}$ may be only a partially ordered list. To be able to apply the aggregation strategy (as we will discuss below), it is necessary to also assign ranks to the candidates not in the list. In our development, given a trial with a partial list, we assign all the uncovered candidates with the same lowest rank. Therefore, in one trial, a covered candidate is always ranked higher than an uncovered one, and two uncovered candidates are equally ranked.

Since essentially any rank aggregation strategy can be applied in our scenario, in our development, we test several different aggregation strategies and our goal is to find the most appropriate one. We first choose the widely deployed Borda's aggregation [10] as the baseline aggregation strategy. We then realize that to enforce the majority voting, it is important that an aggregation strategy satisfies the *Condorcet criterion* [69]. We thus propose a strategy, *FK aggregation*, by combining Kemeny optimal aggregation [44] and footrule optimal aggregation [29]. We will discuss these two strategies, *i.e.*, Borda's aggregation and FK aggregation, in detail respectively.

Baseline: Borda Aggregation: A primary strength of Borda's aggregation is that it is rather simple and computationally efficient: It can be implemented in linear time. Borda's aggregation also satisfies some good properties such as anonymity, neutrality, and consistency [68]. Specifically, in Borda's aggregation, given a candidate M_j, let r_{ji} be the number of matchings ranked lower than M_j in $PR^A_{I_i(S)}$,

the *borda score* of M_j, denoted as $B(M_j)$, is defined as the sum of all r_{ji}, *i.e.*, $B(M_j) = \sum_{k=1}^{T} r_{jk}$. The aggregation result $\mathbb{R}^A_{\mathbb{I}(S,T)}$ is thus the descending ordering of all the candidates with respect to their borda scores.

Example 22: Continue on Example 21, after candidate selection, we first complete the partial lists. In particular, since $PR^A_{I_2(S)}$ only partially ranks the four candidates, we assign the lowest rank to the uncovered candidate M_4, *i.e.*, we rank M_4 as the $4th$ candidate in $PR^A_{I_2(S)}$. Next, we compute the borda score for each candidate and then apply Borda's aggregation. In particular, since M_1 is ranked higher than 3 candidates in $PR^A_{I_1(S)}$, 2 in $PR^A_{I_2(S)}$ and 2 in $PR^A_{I_3(S)}$, the borda score for M_1 is 3 + 2 + 2 = 7. Similarly, the borda scores for M_2 to M_4 are 6, 5, 0 respectively. The final ranking $\mathbb{R}^A_{\mathbb{I}(S,T)}$ is thus $M_1 > M_2 > M_3 > M_4$.

Enforcing Majority by Satisfying the Condorcet Criterion: FK Aggregation: Our analysis of the effectiveness of the ensemble DCM framework in Section 4.2 is based on the assumption that when a matching is correctly ranked in the majority of trials, it will be correctly ranked in $\mathbb{R}^A_{\mathbb{I}(S,T)}$. Therefore, our aggregation strategy should reflect this requirement of majority– That is, if a matching can be correctly ranked in most trials, its ranking in $\mathbb{R}^A_{\mathbb{I}(S,T)}$ should also be correct.

We notice that this requirement is consistent with the classic *Condorcet criterion* [69]. Specifically, the Condorcet criterion requires that, given any two candidates M_i and M_j, if a majority of trials ranks M_i higher than M_j, then M_i should be ranked higher than M_j in the aggregate list $\mathbb{R}^A_{\mathbb{I}(S,T)}$. (As we can see here, setting the number of trials T as an odd number, as Section 4.2 discussed, can ensure that there will be no tie situation between any two M_i and M_j.) The fact that aggregation mechanisms that satisfy the Condorcet criterion can yield robust results has also been noticed and exploited by [29]. However, Borda's aggregation, although computationally very fast, does not satisfy the Condorcet criterion. To our knowledge, the only aggregation strategy exactly satisfies the Condorcet criterion is Kemeny optimal

aggregation. Another strategy, footrule optimal aggregation, does not directly satisfy the Condorcet criterion, but its ordering of matchings yields a factor-2 approximation to Kemeny optimal aggregation.

Example 23: To see how Borda's aggregation may not satisfy the Condorcet criterion, let us see an example, which is slightly different from Example 21. Assume after candidate selection, we have $RR^A_{I_1(S)}$: $M_1 > M_2 > M_3 > M_4$, $PR^A_{I_2(S)}$: $M_1 > M_2 > M_4 > M_3$, and $PR^A_{I_3(S)}$: $M_2 > M_3 > M_4$.

With Borda's aggregation, we have the borda scores for M_1, M_2, M_3 and M_4 as 6, 7, 3, 2 respectively. The ranking of matchings under Borda's aggregation is thus $M_2 > M_1 > M_3 > M_4$. However, M_1 is ranked higher than M_2 in the majority of trials, *i.e.*, $RR^A_{I_1(S)}$ and $RR^A_{I_2(S)}$, which shows that Borda's aggregation violates the Condorcet criterion and therefore may not reflect the results of majority.

Although Kemeny optimal aggregation satisfies the Condorcet criterion, it is computationally expensive. Kemeny optimal aggregation is to find the ordered list $\mathbb{R}^A_{\mathbb{I}(S,T)}$ that minimizes $\sum_{i=1}^{T} K(PR^A_{I_i(S)}, \mathbb{R}^A_{\mathbb{I}(S,T)})$, where $K(PR^A_{I_i(S)}, \mathbb{R}^A_{\mathbb{I}(S,T)})$ denotes the *Kendall tau* distance. That is, it is the number of pairs of candidates (M_i, M_j) on which the ordered lists $PR^A_{I_i(S)}$ and $\mathbb{R}^A_{\mathbb{I}(S,T)}$ disagree (*i.e.*, one ranks M_i higher than M_j, while another one ranks M_j higher than M_i). It has been proven that computing Kemeny optimal aggregation is NP-Hard [29], which is not affordable in practice. Hence, we cannot only apply this aggregation strategy.

As the approximation to Kemeny optimal aggregation, footrule optimal aggregation has good computational complexity. In footrule optimal aggregation, the aggregate list $\mathbb{R}^A_{\mathbb{I}(S,T)}$ contains the median ranks of all the matchings. Specifically, given a candidate M_j, let q_{ji} be the rank of M_j in $PR^A_{I_i(S)}$, the *median rank* of M_i is defined as $median(M_j) = median(q_{j1}, ..., q_{jT})$. The aggregation result $\mathbb{R}^A_{\mathbb{I}(S,T)}$ is thus the ordered list of median ranks of all the candidates. Footrule optimal aggregation can be computed in polynomial time. Although it may not satisfy the Condorcet criterion, it has been shown that its ordering of matchings (*i.e.*, the footrule distance) has a factor-2 approximation to the Kendall tau

distance in Kemeny optimal aggregation [25]. However, footrule optimal aggregation suffers the tie problem. That is, some matchings may have the same median rank and it is unclear how to break ties in footrule optimal aggregation.

Combining the strength of these two aggregation strategies, in our development, we develop a hybrid aggregation strategy, *FK aggregation*. In particular, we first apply footrule optimal aggregation. To break a tie, we apply Kemeny optimal aggregation only locally for ranking the candidates that cause the tie. Empirically, since the number of candidates result in a tie is often very few (*e.g.*, less than 4), the computation is very efficient.

Example 24: Let us apply FK aggregation for the case in Example 23. We first complete the partial lists. In particular, since $PR^A_{I_3(S)}$ only partially rank the four candidates, we assign the lowest rank to the uncovered candidate M_1.

We then compute the median rank for each candidate and apply footrule optimal aggregation. In particular, the median rank for M_1 is median(1, 1, 4) = 1. Similarly, the median ranks for M_2 to M_4 are 2, 3, 3 respectively.

Since M_3 and M_4 get a tie in footrule optimal aggregation, we break the tie by applying Kemeny optimal aggregation only on M_3 and M_4. Since two out of the three trials prefer M_3 than M_4, we rank M_3 higher than M_4. The final ranking $\mathbb{R}^A_{\mathbb{I}(S,T)}$ is thus $M_1 > M_2 > M_3 > M_4$, which is consistent with the result of only applying Kemeny optimal aggregation, but more efficient.

4.5 Experiments

We evaluate the ensemble DCM approach over real query interfaces. In particular, we implement all the algorithms in Python 2.4 and test all the experiments on a Windows XP machine with Pentium M

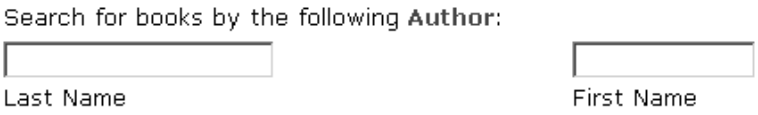


Figure 4.4: An example of incorrectly extracted query interfaces.

1.6GHz CPU and 512M memory. We use two representative domains, Books and Airfares, in the TEL-8 dataset of the UIUC Web integration repository [17] as the testbed.

Our experiments is to verify the impact of noise in the interface extraction on our matching algorithm and evaluate the performance of the ensemble approach. In particular, we conduct our evaluations on automatically extracted interfaces in two domains: Books and Airfares. First, we directly run the base DCM framework on automatically extracted interfaces as the baseline result that we will compare to. Second, we measure the accuracy of the ensemble DCM framework and compare it to the baseline result. The experiments show that the ensemble approach can significantly improve the matching accuracy of DCM. Third, we execute the ensemble DCM framework under various parameter settings and compare the empirical values with our theoretical analysis.

Next, we report our experimental results in detail. All the experiments are conducted with the setting of frequency threshold as 20% (*i.e.*, F = 20%). For more detail about the setting of frequency threshold, please refer to Section 3.5.2 in Chapter 3.

The baseline matching result: The baseline result we will compare to is executing the base DCM algorithm on automatically extracted interfaces. In particular, we use the techniques in [72] to extract interfaces in two domains, Books and Airfares. The second and third columns in Figure 4.5 show the result, where the second column is the target precision and the third column the target recall.

We can see that the accuracies of the baseline approach degrades up to 30%, comparing to the results in Figure 3.10. This performance degradation is mainly because the existence of noise affects the qualification and ranking of matchings and thus the result of matching selection. For instance, in

Domain	*The base DCM framework*		*The ensemble DCM framework* with Borda's aggregation				*The ensemble DCM framework* with FK aggregation			
	P_T	R_T	P_{AT}	R_{AT}	P_{FT}	R_{FT}	P_{AT}	R_{AT}	P_{FT}	R_{FT}
Books	0.73	0.75	0.83	0.89	0.9	1.0	0.83	0.9	0.9	1.0
Airfares	0.67	0.68	0.79	0.79	0.71	0.82	0.91	0.73	1.0	0.73

Figure 4.5: The comparison of target accuracy on Books and Airfares.

the Books domain, author = last name is ranked higher than author = {last name, first name} because in some interfaces (*e.g.*, the ones shown in Figure 4.4), the input box which should be associated with "Last Name" is incorrectly associated with "Search for books by the following Author". Such errors lower down the negative correlation between author and first name and thus result in the selection of the partially correct matching author = last name.

Also, due to the greedy selection strategy, the errors caused in one iteration may cascade to its subsequent iterations. For instance, still in the Books domain, when author = {last name, first name} is pruned out (because of the above reason), in the next iteration of selection, isbn = {last name, first name} is selected as a correct matching, which makes the result even worse.

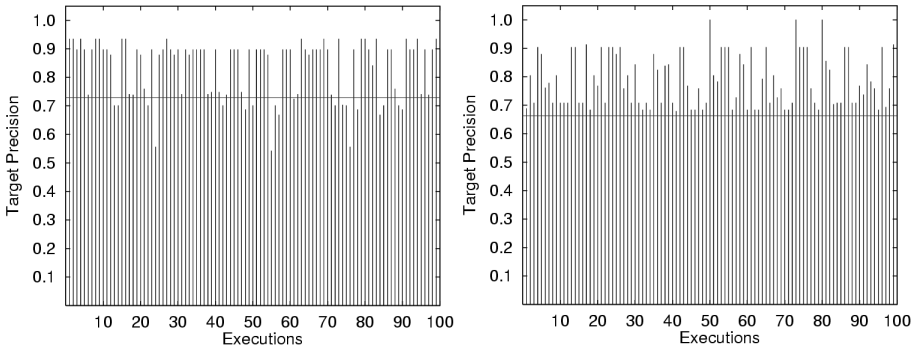
The performance of the ensemble DCM framework: Before running the ensemble framework, we need to first determine its configuration. In our experiment, we choose the $T \rightarrow S$ configuration strategy developed in Section 4.3. Specifically, we set the number of trials T as 41 and objective voting confidence c as 0.9 for both Books and Airfares. (As we modeled in Section 4.2, T is set as an odd number. We have no particular reason for choosing 41. As Section 4.3 discussed, S is insensitive to T and thus picking other T values will not significantly affect the final performance. We also empirically verify this fact later.) We then set W^* and K^* values according to our estimation strategy of the base parameters. In particular, for Books, we have $W^* = 6$ and for Airfares, $W^* = 5$. For both domains, we set K^* as a small constant 2. Thus, according to Equation 4.6, we have $S = 22$ for Books and $S = 19$ for Airfares. Also,

for each dataset, we test it with the two aggregation strategies we developed in Section 4.4 respectively: The Borda's aggregation and the FK aggregation.

As the ensemble framework is essentially a data-randomized approach (with multiple random trials), it is "non-deterministic"– We thus measure the distribution of its performance. Specifically, we execute the framework 100 times on Books with the same setting S = 22, T = 41. Similarly, we execute it 100 times on Airfares with the same setting S = 19, T = 41. To quantify the comparison with the baseline result, we measure two suites of target accuracies: the *average target accuracy* (*i.e.*, the average precision and recall of the 100 executions, denoted as P_{AT} and R_{AT} respectively) and the *most frequent target accuracy* (*i.e.*, the most frequently obtained precision and recall of the 100 executions, denoted as P_{FT} and R_{FT} respectively). Note that we do not use the best target accuracy (*i.e.*, the best precision and recall of the 100 executions) because in practice we cannot judge which result is the best without knowledge from human experts. In contrast, most frequent accuracy is more meaningful since it can be obtained by executing the ensemble framework multiple times and taking their majority.

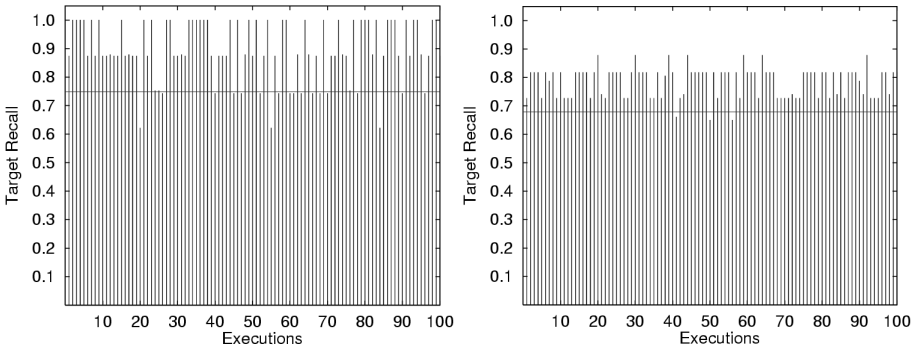
The results of both average and most frequent accuracies are listed in Figure 4.5 (columns 3-6 for Borda's aggregation and columns 7-10 for FK aggregation). We can see that: 1) Comparing to the baseline result, precision and recall are improved by the ensemble framework under both aggregation strategies. 2) For the Books domain, Borda's aggregation and FK aggregation have roughly the same accuracy; For the Airfares domain, FK aggregation can achieve much higher precision than Borda's aggregation, but with slightly lower recall.

Overall, the ensemble framework is quite effective in maintaining the robustness of the DCM matcher. The FK aggregation strategy can yield more robust results than Borda's aggregation. We believe this experiment shows that, while Borda is actually a reasonable baseline choice, FK is indeed more robust. Next, we illustrate and interpret the results of the ensemble framework with more detail.



(a) Books. **(b)** Airfares.

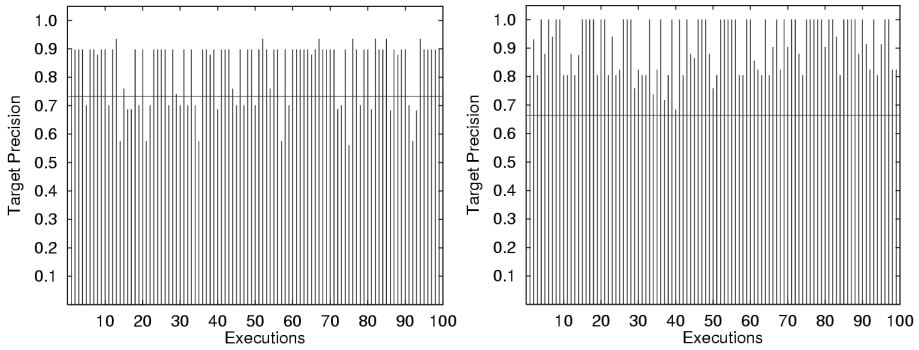
Figure 4.6: The target precision with 100 executions on two domains (Borda's aggregation).



(a) Books. **(b)** Airfares.

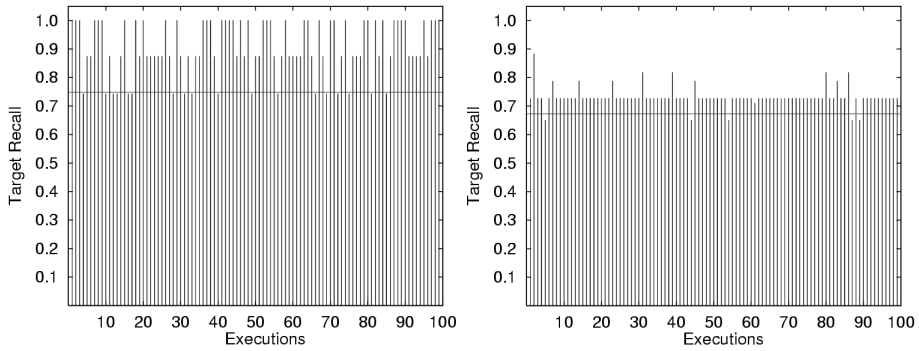
Figure 4.7: The target recall with 100 executions on two domains (Borda's aggregation).

First, in most executions, the ensemble framework achieves better accuracy than the baseline result. For instance, Figure 4.6 shows the 100 target precisions of the 100 executions over Books and Airfares with Borda's aggregation. To make Figure 4.6 more illustrative, we use straight horizontal lines to denote the baseline accuracies. We can see that, although accuracies may be varying in different executions, most precisions in both Books and Airfares are better than their corresponding baseline precisions. Similar result can also be observed in the target recall part (Figure 4.7) under Borda's aggregation and both precision (Figure 4.9) and recall (Figure 4.9) under FK aggregation. Hence, this experiment indicates that the ensemble framework can indeed boost the matching accuracy under noisy schema input, and thus maintain the desired robustness of a holistic matcher. Note that the recall graphs looks more



(a) Books. **(b)** Airfares.

Figure 4.8: The target precision with 100 executions on two domains (FK aggregation).



(a) Books. **(b)** Airfares.

Figure 4.9: The target recall with 100 executions on two domains (FK aggregation).

regular than the precision ones because for recall, only the value on numerator is changing, while for precision, values on both numerator and denominator are changing.

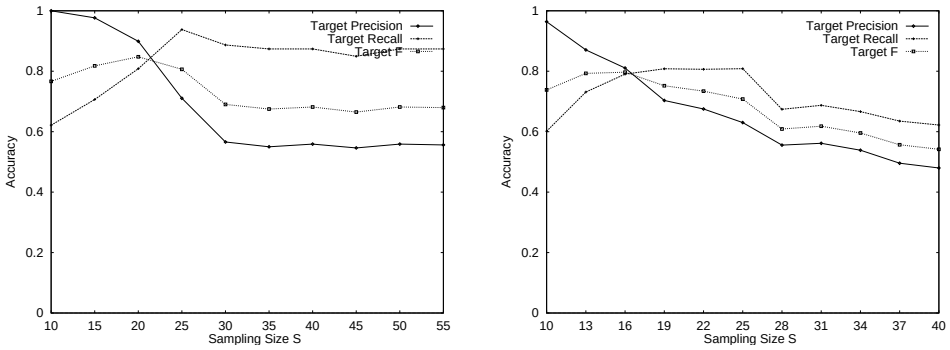
Second, from Figures 4.6 to 4.9, we also observe an interesting phenomenon: It seems that there are upper-bounds for both precision and recall, which the ensemble framework cannot exceed. The existence of such upper bounds is because, in essence, there are two types of data quality problems, noise and missing data, and the ensemble framework can deal with noise, but not missing data. Specifically, noise refers to some observed data that ideally should not be observed, *i.e.*, *outliers*. For instance, the extraction of a book schema, *e.g.*, the one in Figure 4.4, may incorrectly consider "author" as an attribute and thus lowers down the correlation of "author" and "first name." Although noise may affect

the accuracy of the base algorithm, they are minority in quantity. Downsampling is thus a good approach to filtering them out and, consequently, the majority voting can be effective. On the other hand, *missing data* are some data that ideally should be observed, but in reality are not. For instance, the input schemas may contain only a small number of occurrences of the attribute "last name" and thus we cannot sufficiently identify to find the grouping of "last name" and "first name." For this missing data case, sampling and voting techniques will not help, since when the entire dataset has missing data, all the trials will also have missing data and their aggregate result cannot fix the problem. The ensemble framework, with the limit imposed by such missing data, has an upper bound for the best accuracy.

Finally, the execution time of the ensemble framework is also acceptable. The 100 executions on Books take 118 seconds for Borda's aggregation and 117 seconds for FK aggregation. The 100 executions on Airfares take 109 seconds for Borda's aggregation and 128 seconds for FK aggregation. Therefore, the average time for one execution is about only 1 second.

The result under various configuration settings: The purpose of this set of experiments is to empirically verify our analysis in Section 4.3: 1) We want to verify whether our setting of S using Equation 4.6 is consistent with empirical observation. 2) We want to verify whether the performance of the framework is indeed insensitive to T, but sensitive to S.

First, we measure the accuracy of the ensemble framework with different sampling sizes on the two domains. In particular, we fix T at 41 and let S progressively increase from 10 to 55 with an increment size 5 (*i.e.*, 10, 15, 20, ..., 55) for Books and from 10 to 40 with an increment size 3 for Airfares. For each sampling size, we execute the ensemble framework 30 times under the two aggregation strategies respectively and compute the average precisions and recalls. Figure 4.10 shows the experimental result under Borda's aggregation and Figure 4.11 FK aggregation.

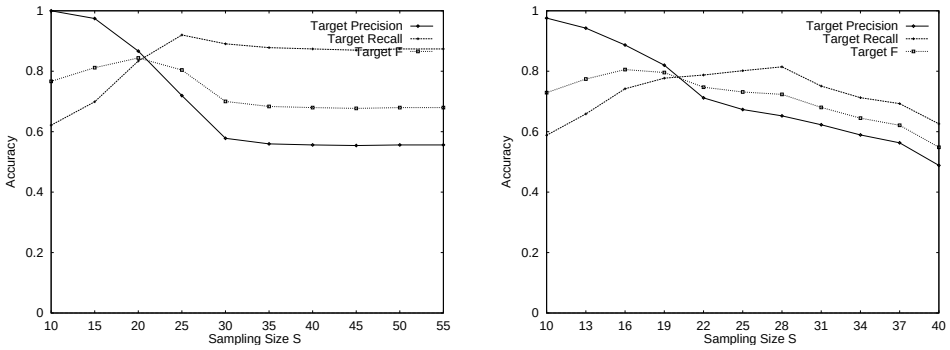


(a) Books (T=41). **(b)** Airfares (T=41).

Figure 4.10: The target accuracy under various sampling sizes (Borda's aggregation).

From Figures 4.10 and 4.11, we can observe the same trend in both domains, which seems to be independent of the aggregation strategy we choose. Specifically, when sampling size increases, the target precision mostly keeps on decreasing, while the target recall goes up first and then goes down at some point. We give the explanation as below: A small sampling size may miss some attributes in downsampling and thus discover less matchings, which results in trivially high precision but low recall. With larger sampling size, we are able to cover more attributes and thus discover not only more correct matchings, but also a few false matchings. Consequently, the precision decreases and recall increases. When the sampling size is too large, a downsampling is likely to have a lot of noise and thus the recall starts to decrease again.

The best sampling size we should take is thus some values in the middle. We choose the F-measure, which combines precision P and recall R as $F = \frac{2PR}{P+R}$, to measure the overall accuracy. From Figure 4.10(a) and Figure 4.11(a), we can see the best range of sampling size for Books, according to F-measure, is around 20. Our setting based on Equation 4.6 is 22, which is quite close to 20. Similarly, from Figure 4.10(b) and Figure 4.11(b), the best range of sampling size for Airfares is around 16. Our setting based on Equation 4.5 is 19, which is also close. Therefore, our configuration strategy of determining the sampling size is consistent with the empirical result.



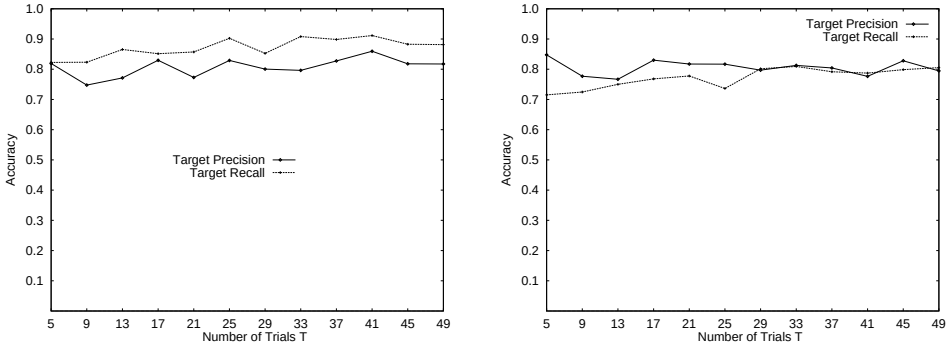
(a) Books (T=41). **(b)** Airfares (T=41).

Figure 4.11: The target accuracy under various sampling sizes (FK aggregation).

Second, since we choose T as 41 with no particular reason in the experiment, we want to verify that the choosing other T values is in fact not quite different, because of the insensitivity of S on T (Section 4.3). In particular, we fix S at 22 for Books and 19 for Airfares. We change T from 5 to 49 with increment size 4 for both domains. For each T, we again execute the framework 30 times under the two aggregation strategies and compute the average precisions and recalls. Figures 4.12 and 4.13 shows the experimental results. From the results, we can see that, in both domains, both the precision and recall become more and more flat and stable when T increases. This result indicates that with other T values (as long as it is not too small), we can also have roughly the same performance and thus the decision on T is not a critical factor.

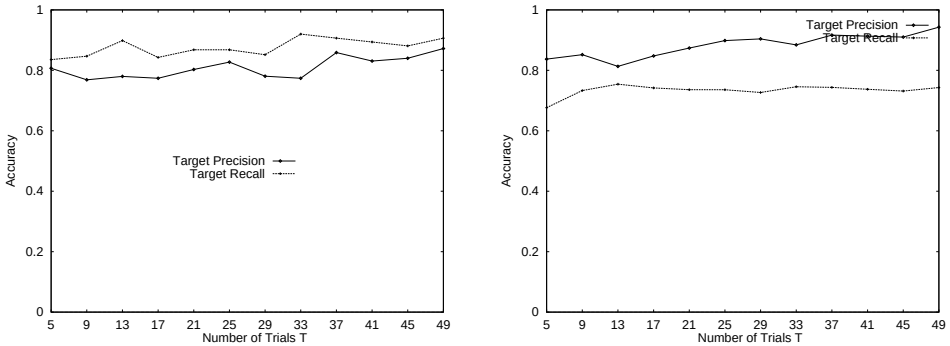
Also, comparing Figure 4.12 and Figure 4.13, we can observe that the ensemble framwork with FK aggregation generally can achieve better precision than the one with Borda's aggregation. This result indicates that FK aggregation is more robust than Borda's aggregation in dealing with noisy data, since it approximates the Condorcet criterion (Section 4.4).

Overall, from these two experiments on S and T, we can see that under the same T, different sampling sizes S will significantly affect the performance of the data-ensemble framework, while on the other hand, under the same S, different number of trials T have little impact on the performance. This



(a) Books (S=22). **(b)** Airfares (S=19).

Figure 4.12: The target accuracy under various number of trials (Borda's aggregation).



(a) Books (S=22). **(b)** Airfares (S=19).

Figure 4.13: The target accuracy under various number of trials (FK aggregation).

sensitivity of performance on S but not T indicates that the $T \rightarrow S$ configuration strategy is better than $S \rightarrow T$ because T is much easier to pick in practice, which verifies our analysis in Section 4.3.

4.6 Conclusion

This chapter identifies robust quality as an inherent challenge for leveraging holistic quantity in large scale schema matching. Such a robustness issue inevitably arises in integrating holistic schema matching with automatic schema extraction. As the solution, we develop an ensemble scheme with sampling and voting techniques, inspired by bagging predictors. We are essentially applying bagging techniques in a

new scenario of mining semantic correspondences among attributes. Both the analytic justification and experimental result show the promise of our framework.

Since our matching algorithms require the input schemas (*i.e.*, query interfaces) from the same domain, to enable such large scale matching, we need to develop automatic techniques to discover query interfaces on the Web (*i.e.*, the source discovery problem) and cluster them into their domain hierarchy (*i.e.*, the schema clustering problem). Chapter 5 and Chapter 6 will discuss our solutions to these two issues respectively.

Chapter 5

Automatic Discovery of Query Interfaces

To enable our matching work, the very first step is to collect a set of query interfaces (in various topic domains). As query interfaces are sparsely scattered on the Web, it is challenging to develop effective crawling techniques to discover query forms in both efficient and comprehensive manners: First, because of the topic-neutral nature of our crawling goal, we cannot rely on existing topic-focused crawling techniques. Second, traditional page-based crawling techniques cannot achieve a good balance between crawling harvest and coverage. To tackle this problem, we develop a *Web Form Crawler* with a new *structure-driven* crawling framework. In particular, we observe *structure locality* of query forms. That is, query forms are often close to root pages of Web sites and accessible by following navigational links. Exploring this structure locality, we substantiate the structure-driven crawling framework into a *site-based* Web Form Crawler by first collecting the site entrances and then searching for query forms within the scope of each site.

5.1 Motivation: Object-Focused Crawling

Building the Web Form Crawler brings new challenges. In particular, while a large number, query interfaces as scattered on the entire Web are rather sparse: Our estimated 1,258,000 query interfaces (as just mentioned; in [16]) can appear anywhere in the 19.2 billion Web pages (as reported by the recent index of Yahoo.com [67], which thus lower bounds the Web size). As a baseline, the traditional *page-based* crawler (without a topic focus), which recursively follow links to traverse the entire Web, will thus expect to find only one interface in crawling 15262 pages.

The task of Web form crawling is thus, literally, searching for a needle in a haystack. To effectively build a database of online databases, as our "map" to the deep Web, our crawler has dual requirements: First, to be *efficient*, it must have a high *harvest* rate, defined as $\frac{\#forms-collected}{\#pages-crawled}$, to collect Web forms without crawling many pages. Second, to be *comprehensive*, it must have a high *coverage* rate, defined as $\frac{\#forms-collected}{\#total-forms}$, so as to cover a reasonable snapshot of the deep Web. To motivate, the traditional page-base crawler, as just mentioned, after crawling the entire Web (or c% pages), will in principle result in 100% (or proportionally c%) coverage, but at the cost of a measly harvest of 6.6×10^{-5}.

For a more effective crawling, instead of traversing arbitrary links, we must develop a *focused* crawling strategy tailored for finding query forms. Unlike general crawling (which generally collects all Web pages), a focused crawler targets a specific subset of pages on certain focus, say, "virtual reality." Such a focused crawler, with its specific target, can often find crawling paths of certain patterns that lead to the desired pages, and thus achieve higher harvest. As our goal is to build a focused Web Form Crawler, we must address two new challenges:

First, our crawler is *object-focused* but topic-neutral– the opposite of traditional focused crawlers. That is, unlike existing settings of *topic-focused* crawlers [15, 27, 54], which look for Web *pages* of certain *topics*, our crawler targets at a certain type of *objects*, namely query forms, which can be of any

subject topics (*e.g.*, Amy's example: real-estate, cars, jobs). With their topic-focus, existing focused crawlers are mainly *content-driven*, by exploiting *content locality* across links: A page of certain topics can often be reached through a path along which the *contents* of pages form some patterns. In the simplest form, such content locality means that a page on, say, "virtual reality" may be connected from pages of similar topics. At its core, a topic-focused crawler employs a classifier to distinguish the content orientations (*e.g.*, trained using keyword features) of pages to find a desirable path. Such techniques, by assuming topic-focus and thus content-driven, are unlikely to work for our object-focused but topic-neutral crawler.

Second, we aim at balancing both efficiency and comprehensiveness, with not only a high harvest but also a "reasonable" coverage. (With the ever expanding and changing Web, it is well accepted that 100% coverage is unrealistic.) We note that harvest and coverage are often *conflicting* metrics: While focusing on only promising pages will lead to a high harvest, its narrow focus of "not going beyond" may compromise the coverage. On the other hand, although combing through many pages will extend the coverage, the broad reach may lead to diminishing returns and thus compromise the harvest. In particular, most existing focused crawlers, by greedily pursuing promising paths, aim at high harvest with no explicit notion of coverage. That is, while a crawler may start with high harvest for what it *has* crawled, how long will such harvest sustain? Can it estimate the harvest for what it has *not* crawled, so as to bail out without wasting resources in diminishing returns that will not enhance coverage (but actually hurt harvest)? With this sense for the "unexplored" territory, it can focus resource on achieving reasonable coverage while maintaining high harvest *throughout* crawling.

Overall, our goal is to develop a crawling framework that will, without assuming topic-focus, not only give a high harvest for what it has crawled, but also estimate a low yield for what it decides not to crawl, and thus achieve a good coverage overall. Our insight hinges on that, for our object-focused

crawling, there exists certain *structure locality* on the Web, which can guide a "scope" for our crawling to focus into and draw a boundary around. This concept of structure locality, in terms of how our target objects distributes in the scope, will enable the dual goals of harvest and coverage.

Specifically, we observe that query forms indeed distribute with such structure locality: *First*, independent of topic domains, query forms often appear near the entrance point, *i.e.*, the root page, of a Web site. Second, around the entrance point of a site, query forms also distribute in certain ways– Within the site, they tend to appear shallowly and are often reachable through *navigational links* (*i.e.*, links in the navigational menus of the site). Thus, our topic-neutral crawler can focus on such structure locality: Viewing the Web as a graph of Web sites, it will crawl each site as a separate "scope." For each site, starting from its entrance and following navigational links, it will achieve a high harvest rate. Further, drilling deeper into the site, when the yield starts to diminish, it will bail out, while still maintaining satisfactory coverage. Iterating over sites, as each site gives good *local* harvest and coverage, our crawling will maintain the same *global* harvest and steadily growing coverage throughout. Finally, since our crawling assumes Web sites as independent "scopes," it is inherently parallelizable.

We thus propose the new concept of *structure-driven* crawling for realizing our *object-focused* crawler. It consists of, conceptually, two phases: Phase 1 continuously finds new scopes, *i.e.*, entry points (or root pages) into a site, and Phase 2 searches for query forms in each site. We construct the *Web Form Crawler* with two components: *Site Finder* for collecting Web sites as scopes, and *Form Finder* for in-site searching each scope. Our conceptual analysis shows that the structural-driven framework will enable balancing of harvest and coverage throughout crawling, which can be configured with different in-site search strategies. We will motivate the structure-driven architecture (Section 5.2) and explain the design of the Site Finder (Section 5.3) and Form Finder (Section 5.4). We have implemented the crawler, in a naturally parallel architecture, and deployed on a cluster of about 100 PC nodes. We

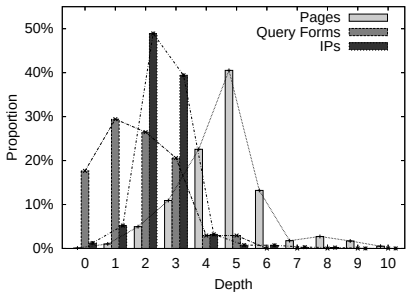


Figure 5.1: Proportion of pages, query forms, and IPs over depth.

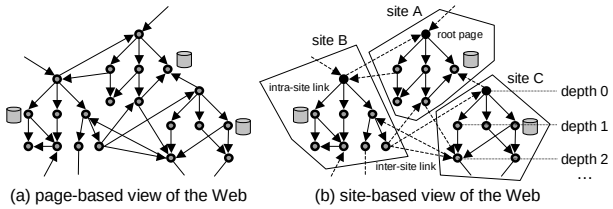


Figure 5.2: Page-based view vs. site-based view.

will report large scale experiments in Section 5.5, which validate that the framework indeed crawls Web forms effectively by maintaining steady harvest and growing coverage as it crawls.

5.2 System Architecture

As Section 1 discussed, we design the Web Form Crawler to realize the structure-driven crawling framework. In this section, We will present our architectural design of the Web Form Crawler in details. Specifically, we first observe the existence of concerted structure locality (Section 5.2.1), then motivate the structure-driven yield-aware crawling framework (Section 5.2.2), and finally discuss the development of the system architecture (Section 5.2.3).

5.2.1 Motivation: Structure Locality

Our object-focused crawling aims at comprehensively collecting query forms as the *target objects*. Being topic-neutral and coverage-aware, unlike traditional topic-focused crawling, our crawling cannot

rely on content locality (as Section 1 mentioned)– We thus wonder, is there any new type of "locality," as distribution patterns of the target objects, that we can resort to?

To know the answer, we attempt to take a "divide-and-conquer" approach for solving the object-focused crawling. We first divide the Web into a set of non-overlapping *scopes*, where each scope contains a unique set of pages. With appropriate partitions, we hope that each scope will contain some type of locality, which can be explored to conquer the problem of object-focused crawling. Then our question becomes: Can we find a good way to divide the Web into scopes with the localities we need for the crawling task?

We notice that Web sites, as the intermediate concept between pages and the entire Web, seem to be natural partitions for scopes. We thus conduct a survey over Web sites and the answer is positive– Our result shows that Web sites are the appropriate scopes with a new locality feature for finding query forms. In particular, we studied the locations of query interfaces in their Web sites. For each query interface, we measured its *depth* as the minimum number of hops from the root page of the site to the interface page. We randomly sampled 1 million IP addresses, from which we identified 281 Web servers, crawled these servers up to depth 10, and identified a total of 34 databases with 129 query interfaces. Since a database can be accessed through multiple query forms in many sites, we manually check all the query interfaces to identify such "same-databases".

Our study shows an interesting phenomenon: Query forms tend to locate "shallowly" in their sites and thus have *structure locality*. Figure 5.1 shows the distribution, in terms of proportion of total query forms, at progressively deeper levels from depth 0 to 10. The result clearly shows that most query forms can be found within depth 3 and none deeper than 5. To contrast in perspective, Figure 5.1 also shows the distribution of pages– which grows *exponentially* from 0 up to 5 and decreases after. While there are significantly more pages toward deeper in a site, most query forms are in the shallow levels. In

particular, the top 3 levels (form the root page to depth 3) contain only 17% of total pages but 94% of forms.

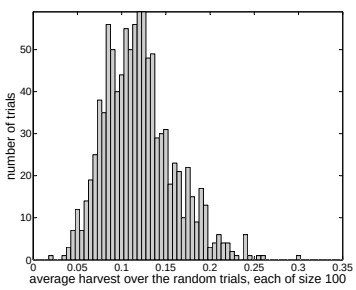
This observation inspires us a site-based view of pages on the Web. To begin with, Figure 5.2(a) shows the typical page-based view of the Web, in which all pages and all links are equal. On top of the page graph, we now view the Web as a collection of Web *sites*, as Figure 5.2(b) shows. Each site is an HTTP server containing a subgraph (of the Web) for pages on the server, and is uniquely addressed by a distinct IP domain name and an HTTP port number [1], *e.g.*, `http://xyz.com:8080`. For brevity, we will simply use *site*, *IP*, or *domain name* interchangeably.

From Figure 5.2, we can see that although query forms seem to distribute sparsely and randomly on the traditional page view, the structure locality as we observed means that, within each site as a scope, the distribution of query forms is rather "predictable" (in a statistical sense)– they follow the pattern as Figure 5.1 shows, in which we expect to find query forms, if any, around the entrance of each scope.

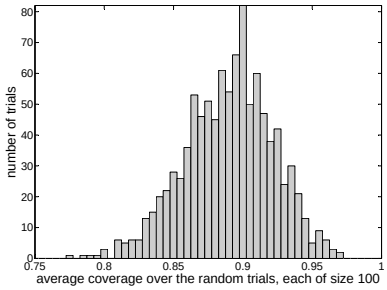
To compare, in the traditional page-based view, we essentially consider each page itself as a scope. Under this view, since each scope is "atomic" with only one page, there is no intra-scope locality. We can only explore the inter-scope locality, *i.e.*, the linkage closeness among scopes (*i.e.*, pages) with the same topic or the so-called content locality.

On the contrary, in the site-based view, we view each site as a scope and employ structure locality as the "maps" to guide the crawling within scopes for finding target objects. Unlike the inter-scope content locality, structure locality, as a new type of locality, explores intra-scope information (*e.g.*, the depth of links, the navigational menu links) and has two excellent features: 1) topic-neutral: By exploring structure information, an intra-scope search strategy can equally handle any scope regardless

[1] With IP aliasing and virtual hosting, there is generally a many-many mapping between IP and domain names. To be precise, a site should be recognized by (domain-name, IP, port-number).



(a) Normal distribution of the mean harvest.



(b) Normal distribution of the mean coverage.

Figure 5.3: Normal distribution of the mean yields.

of its domain. 2) coverage-aware: Equally treating any scope, an intra-scope search strategy is likely to achieve stable harvest and coverage within scopes and further make the overall yields predictable.

5.2.2 Methodology: Structure-Driven Crawling

The observation of the structure locality motivates us a new concept, *structure-driven* crawling, as a framework for building object-focused crawlers. This concept parallels and contrasts the implicit notion of *content-driven* crawling framework behind existing topic-focused crawlers, as Section 1 introduced. In a structure-driven framework, a crawler conceptually partitions the Web into independent scopes and searches for target objects in each scope, with certain intra-scope search strategy that matches the object distribution patterns. If such structure-locality patterns indeed exist, the in-scope search strategy can explore different ways to achieve predictable harvest and coverage for the crawling.

For each scope, we crawl from its entrance to search within the scope, guided by an *in-site* search strategy. By matching the structure locality of the scope, different strategies will result in different tradeoff of yield rates in this scope, or *local harvest h* and *local coverage c*%. To confidently predict the "global" harvest *H* and coverage *C*% of the entire crawling process from our local yields, we develop a high-level methodology for structure-driven crawling.

First, *sampling phase*: Suppose we have a set of alternative intra-scope search strategies of crawling a certain type of objects. Our goal in the sampling phase is to select the best strategy by testing the

strategies over a randomly sampled set of scopes [2]. To show that such a sampling phase can indeed help us predict the overall performance, we need to address two issues: 1) We need to show that, by choosing an appropriate sampling size, we can guarantee a confident estimation of the local yields of a strategy. 2) We can predict the global yields from the local yields. We will discuss these two issues respectively in this section.

Second, *executing phase*: We apply the selected strategy for crawling over the whole Web. With the accurate estimation of the local yields based on the sampled scopes, we thus can accurately predict the global yields in this phase. Our empirical study in Section 5.5 shows that the executing phase can indeed maintain steady harvest and coverage in practice.

To illustrate why local yields can be accurately estimated and they can further imply global yields in structure-driven crawling, let us for now consider a scope as a Web site and its entrance as the root page. Assume we use a simple strategy, *Exhaustive*(3), for crawling pages in a site from the root (*i.e.*, depth 0) up to depth 3.

Local Harvest h and Coverage c%: As the same intra-scope crawling strategy may not generate the same local yields for different Web sites, we wonder whether we can observe stable local yields and further estimate them (in a statistical sense) with a randomly sampled set of sites. According to the Central Limit Theorem [9], when the sample size is large (usually more than 30), we can calculate confidence intervals of the mean values of the local harvest and local coverage using Equation 5.1. That is, the mean value μ of a random variable X has $1-\alpha$ probability to be in the range $[\overline{X}_n - \frac{z_{\alpha/2}s}{\sqrt{n}}, \overline{X}_n + \frac{z_{\alpha/2}s}{\sqrt{n}}]$, where n is the sampling size, X_i is the ith sample, $\overline{X}_n = \frac{\sum_{i=1}^{n} X_i}{n}$, and $s^2 = \frac{\sum_{i=1}^{n}(X_i - \overline{X}_n)^2}{n-1}$. For instance, with a trial of sampling 1000 sites, we have that the 95% confidence intervals of the local harvest mean and the local coverage mean of *Exhaustive*(3) are 0.119 ± 0.02 and 0.898 ± 0.016 respectively.

[2]The criterion of judging the best is specific to the crawling task. For instance, a possible criterion can be choosing the strategy with the highest harvest among all strategies satisfying a given coverage.

$$P(\overline{X}_n - \frac{z_{\alpha/2}S}{\sqrt{n}} < \mu < \overline{X}_n + \frac{z_{\alpha/2}S}{\sqrt{n}}) \quad = \quad 1-\alpha \tag{5.1}$$

To visually illustrate the above estimation of confidence intervals, we conduct 1000 trials, with each trial sampling a different set of 100 sites. We still use *Exhaustive*(3) as the intra-scope strategy to crawl each site. We then compute the average harvest and coverage for each trial and draw the distribution of the average harvest and coverage among the 1000 trials, as Figure 5.3 shows. We can clearly observe that the mean values of both local harvest and coverage show normal distributions, with most values (about 95%) falling into the estimated confidence intervals.

In practice, if we feel the confidence interval we get is not convincing enough to estimate the local yields, we can enlarge the sampling size. According to Equation 5.1, by doing so, we can obtain a smaller confidence interval for the same confidence $1-\alpha$ and thus a better estimation. From our experience, Web sites tend to share the structure locality shown in Figure 5.1. Therefore, we can often achieve accurate estimation of local harvest and coverage with a relatively small sampling size. Our experiment in Section 5.5 will empirical verify this argument for a set of different intra-scope crawling strategies, *e.g.*, the approaches we developed in Section 5.4.

Global Harvest H and Coverage C%: How does the local performance imply the global yields in the entire crawling? We build a simple analytical model for this "prediction": We assume that query forms can only have duplicates in the same site (*e.g.*, Amazon.com has "product search" repeated in every page), and we consider forms from different sites as distinct (thus a form at Amazon.com cannot be found at BN.com). Suppose the following characteristic parameters: 1) There are totally n Web sites.

2) In average, each site has m query forms. 3) The in-site searcher can achieve local harvest h and local coverage c% in average in a site.

We can now derive the global performance: During the entire crawling, the crawler will find $n \times m \times c$%, among the total $n \times m$ forms. The number of pages crawled is $\frac{n \times m \times c\%}{h}$, since it crawls h times more pages than forms. Thus:

$$\text{Global harvest } H = \frac{n \times m \times c\%}{\frac{n \times m \times c\%}{h}} = h \tag{5.2}$$

$$\text{Global coverage } C\% = \frac{n \times m \times c\%}{n \times m} = c\% \tag{5.3}$$

Although it is possible to build more sophisticated modeling to more accurately capture the relationship between local yields and global yields, our empirical study in Section 5.5 shows that the simple modeling is pretty good in predicting the global yields.

Overall, we thus observe two desirable properties entailed by structure-driven crawling:

- **Steady local yields and predictable global yields:** In structure-driven crawling, we can accurately estimate the steady local harvest and coverage with an appropriate sampling size. Such steady local yields will help us to predict the global performance. Such features cannot be supported by traditional topic-focused crawling. Our experiments in Section 5.5 also empirically verify this analysis.

- **Yield-guided crawler design:** The analytical model, while simple, can guide the design of intra-scope search strategies (*e.g.*, selecting d for *Exhaustive*(d)). Guided by a desirable H and C%, we can pick an intra-scope strategy that generates the corresponding local h and c%, thus allowing a principled way of harvest-coverage tradeoff and resource allocation.

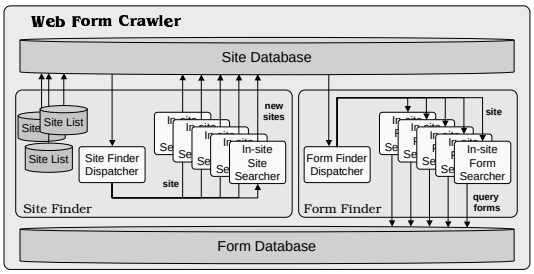


Figure 5.4: System architecture of the Web Form Crawler.

Although our analyses above assume Web sites as the scopes, we believe they are generally applicable to other scope definitions as long as some topic-neutral structure locality can be found within scopes.

5.2.3 Implementation: Architectural Design

To realize the structure-driven crawling framework for collecting query forms, we develop the *Web Form Crawler*. Our crawler conceptually takes a site-based view of the Web, as Figure 5.2(b) shows. In this view, each site is an independent scope, with an expected structure locality of our target objects. (With further analysis, more refined patterns can be constructed; Section 5.4.) We thus substantiate the concept of structure-driven crawling into a *site-based* framework: Crawl each site as a scope, with an *in-site* strategy that matches the locality, to achieve the objective of having predictable harvest and coverage.

Figure 5.4 shows the architecture of the Web Form Crawler, with a pair of concurrent components: *Site Finder* for finding site entrances and *Form Finder* for searching each site for query forms. In this site-based framework, the finding of site IPs and the subsequent in-site search will run concurrently. In particular, the Site Finder collects new site IPs into a *Site Database*. From there, the Form Finder then continuously gets a site entrance, searches for query forms, and collects them into a *Form Database*– the end product of crawling.

Site Finding: To begin with, we need to find a set of sites in terms of entrance IP addresses. There are multiple ways to collect site IP entrances. First, *directory databases*: Some Web directories provide pre-compiled site lists. For instance, DMOZ has compiled a list containing 860,000 sites [57]. Second, *piggyback crawling*: We can add site-IP discovery as a "side effect" of other crawling activities. In particular, as our crawler searches for query forms in-site, it can also "piggyback" IP entrances found alone the way as byproducts. Third, *dynamic discovery*: We can build a crawler to specifically search for site entrances from Web pages. Our Site Finder currently supports two ways of collecting sites: from directory databases (*e.g.*, DMOZ) and from dynamic discovery.

Although our framework will employ multiple means of site finding, with the changing and expanding nature of the Web, we believe dynamic discovery remains essential for covering comprehensive all site-IPs. As a support, Section 5.5 will compare our dynamically discovered IPs with the DMOZ list, which reveals that such (manually) compiled list can be rather limited.

Such dynamic discovery of IPs turns out to be itself an object-focused crawling task (with "sites" as target objects) and can thus also be realized by structure-driven techniques. We observe the same phenomenon as query forms– That is, pages containing new site IPs are close to root pages. We randomly select 100 sites from the DMOZ site list and crawl each site up to depth 10. (We name this dataset as *Random100*, which will be used throughout this chapter.) We measure the distribution at each depth, as Figure 5.1 also shows. The result indicates that structure locality indeed exists: 95% IPs can be found within depth 3. (Section 5.3 will further refine the locality.) Similar to finding query forms, we can resort to structure-driven crawling for site finding. Therefore, while our site-based crawling framework relies on the function of site finding, this function, recursively, can be realized in the same site-based framework.

The Site Finder thus shares the same design as the Form Finder: Within the Site Finder, we schedule site IPs (that are already in Site Database) to search; for each site, we devise an in-site searcher. Hence, our discussion next on site scheduling and in-site search are applicable for both the Form Finder and the Site Finder. (Their different in-site strategies will be explored in Sections 5.4 and 5.3).

Site Scheduling: After collecting sites as scopes, we must develop a scheduling strategy, to order these scopes for in-site search of query forms. There are various alternatives in scheduling: To begin with, *simple iteration* orders all sites arbitrarily, and crawls each till completion. This scheme requires minimal scheduling logic, but may not optimize for important sites, and may not interleave crawling traffic to a single site. To contrast, we can use *ranked interaction* to prioritize site rankings with estimated importance (*e.g.*, some "PageRank"). Similarly, we can adopt *round-robin iteration* to go in "rounds," each of which crawls progressively larger part into a site (*e.g.*, depth d in round d), and thus interleaves site traffic.

Our implementation currently uses simple iteration, for its simplicity. In particular, our experience shows that the concern of site traffic is not significant: Since we aim at searching each site minimally (by exploiting structure locality), the traffic is often rather minor. We emphasize that, for a given set of sites to crawl, different scheduling strategies will *not* affect the global yield (as Eq. 5.2 and 5.3 show), since in principle we will eventually crawl all the sites.

Specifically, to schedule, the dispatchers in Figure 5.4 send site IPs to the concurrent in-site searchers at parallel machines. Note that, since our structure localities suggest that target objects are connected and reachable from their site entrances, our structure-driven framework will search a scope *independently*, without requiring cross-site communication [20]– parallelization is thus immediate.

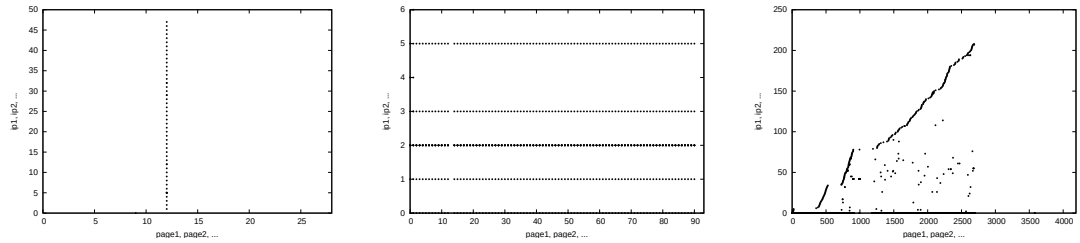
In-Site Search: We develop a generic in-site search logic, which is applicable for both the Form Finder and the Site Finder. As Figure 5.5 outlines: URLs to be crawled are added into a queue Q. In each

```
Algorithm: GENERALINSITESEARCHER:
Input: a site IP ip, maximal depth d
Output: a set of discovered objects from site ip
begin:
1   Q = ∅  /* Q: the queue of urls to be crawled */
2   B = ∅  /* B: blacklist: the set of urls already crawled */
3   I = ∅  /* I: the set of objects found in site ip */
4   Q.enqueue(ip)
5   while Q ≠ ∅
6     /* get a url to crawl and then add it into the blacklist*/
7     url = Q.dequeue()
8     page = retrieve the page of url
9     B = B ∪ {url}
10    /* add new objects in the crawled page page into I */
11    O = OBJECTEXTRACTION(page)
12    I = I ∪ O
13    /* select promising intra-links to crawl */
14    L = LINKSELECTION(page)
15    for each link u ∈ L and u ∉ B ∪ Q and DEPTH(u) ≤ d
16        Q.enqueue(u)
17  return I
end
```

Figure 5.5: Algorithm GENERALINSITESEARCHER.

while-loop, the searcher gets a URL from Q to crawl and extract objects (either site entrances or query forms in our case) from the page by calling OBJECTEXTRACTION. It then selects links to crawl by executing LINKSELECTION and adds these links to Q. The parameter, maximal depth d, controls the depth of crawling. The process terminates when Q is empty.

The function OBJECTEXTRACTION extracts target objects from a page. While this extraction is necessary, it is not our focus in this thesis, and we only briefly explain our implementation: Extracting site IPs is straightforward– We identify inter-site hyperlinks, extract IPs (*e.g.*, `foo.com:8080/abc.html` to `bar.com:8080`), and store them to the Site Database. However, extracting query forms is more involved: For each potential form, as marked by the HTML tag <FORM>, we first decide if it is indeed a query form, to avoid non-interesting forms (for our purpose), *e.g.*, site searches, logins, and polls. We implement the form-detection classifier in [22] for this decision. For each positive form, we then remove



(a) Link-page template. **(b)** Skeleton template. **(c)** Directory template.

Figure 5.6: Three typical templates of the distribution of IPs within a site.

duplicates (by comparing to forms already found in the same site), extract its query structure (by the visual parser in our earlier work [72]), and store it into the Form Database.

Our remaining task is thus to design effective in-site search strategies, *i.e.*, to substantiate the LINKSELECTION function, as guided by the objective harvest and coverage. We study such strategies for the In-Site Form Searcher and the In-Site Site Searcher in Sections 5.4 and 5.3, respectively. The development of such strategies, albeit for different objects, essentially follow the same approach: First, to explore structure locality, we will start with making deeper *observations* to find more structure locality *patterns*. Second, guided by the patterns, we then formulate search *strategies* and provide our specific *implementations*. Such strategies often are configured with parameters (*e.g.*, maximal depth d) which will lead to different harvest and coverage. Finally, we use the *sampling-and-executing* methodology (Section 5.2.2) to select a good intra-scope strategy that leads to desirable performance.

5.3 In-Site Site Searcher

In this section, we discuss specializing the Algorithm GENERALINSITESEARCHER for the task of finding site IPs within a site. In particular, we need to specialize the LINKSELECTION function for selecting links that are likely to contain new sites. Similar to the procedure taken in Section 5.4, we discuss our observations and discovered patterns for the structural locality of site entrances (Section 5.3.1), from which we develop the link selection strategy and implementation (Section 5.3.2).

5.3.1 Observations and Patterns

Observations: We study the occurrences of external site IPs in a site by surveying the 100 sites in the *Random100* dataset. For each site, we draw a "matrix" of IP occurrences. The x-axis is all pages in the site in their breadth first traversal order. The y-axis is all IPs in the site ordered by their first discovery (because an IP can occur in many pages of a site). If an IP occurs in a page, we mark the corresponding position in the matrix with a dot.

From all the occurrence matrices, we observe that the distribution of IPs in a site has three typical shapes, which we call "templates," as Figure 5.6 illustrates. 1) *Link-page template*, in which there is one (or a few) "link" page that contains many external IPs, while other pages have none. 2) *Skeleton template*, in which some external IPs occur in almost every page, mainly because these IPs appear in the common "skeleton" that many pages share. 3) *Directory template*, in which new IPs keep growing and show a triangle shape of distribution– That is, new IPs can occur in not only shallow pages but also deeper ones and thus form a triangle. A site with this template is often a directory site, *e.g.*, Yahoo directory and DMOZ directory, where each page contains IP references for a certain subject category. Finally, some sites may show a mix of these typical templates in their distributions. Among the 100 sites, 11 sites have no new IPs, 44 sites follow only link-page template, 8 only skeleton template and 8 only directory template. There are totally 29 sites that have mixed templates.

Reachable Patterns: From our survey of templates, we summarize two patterns to reach pages containing IPs: First, *target-page pattern*: Some pages are important by themselves. In particular, it is crucial to find the path to the "link" pages. Second, *continuous pattern*: Some sites contain external site IPs in a "continuous" distribution across depths. That is, searching more pages in a site will either continuously find different IPs (*i.e.*, the directory template) or the same ones (*i.e.*, the skeleton template).

5.3.2 Strategy and Implementation

Strategy: We design our link selection strategy by leveraging both reachable patterns. *First*, for sites of the target-page pattern, we observe that link pages are often either close to root page (*i.e.*, within depth 1) or contain some keywords (*e.g.*, "`links`", "`resources`"). We thus design our crawling strategy as: Crawling pages up to depth 1 and then for deeper pages, we build a classifier using anchor-text keywords as distinguishing features to reach special link pages. *Second*, for sites of the continuous pattern, it is clear that we want to leverage the continuity to use the past to "predict" the future and stop early if no more new IPs are found. We thus develop an adaptive crawling strategy, which dynamically decides whether to further crawl or not by its current crawling yields.

Specifically, the continuous distribution of IPs indicates that if we observe enough IPs from a group of recent pages, we are likely to see new IPs in their children. To realize this idea, we need to define what "recent" pages are and how many IPs are "enough." Given a page, we define recent pages as a sliding window of a group of S adjacent intra-site links in the page. For each window, we crawl its pages and compute the number of new IPs found in pages of the window. If the number of new IPs is no lower than a threshold T, we will crawl children pages for every page in the window.

Example 25: To illustrate the adaptive crawling with an example, consider a Web site as Figure 5.7 shows. For simplicity, we alphabetically mark each page, *i.e.*, *a*, *b*, ..., instead of using URLs. Also, for each page, we give it the number of new IPs it yields. Suppose we set the window size S as 2 and the IP threshold T as 5.

To begin with, we crawl the root page *a* and find 6 new IPs[3]. Since 6 is more than the threshold 5, we will continue to crawl its children pages in depth 1, *i.e.*, *b*, *c*, *d* and *e*.

[3]Note that this example is just for illustration. As we will show in Algorithm ADAPTIVE, in practice, to avoid missing the entire site due to a low harvest rate at a single root page, we crawl all pages up to depth 1 regardless of the harvest rate of the root page.

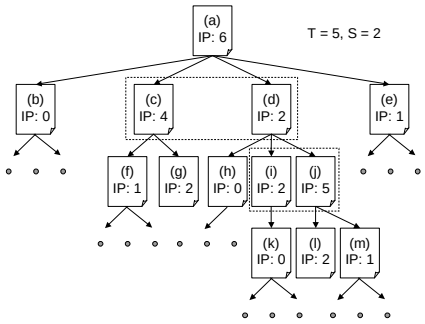


Figure 5.7: Adaptive crawling for finding IPs.

Then, we use sliding windows to decide whether to crawl pages in depth 2 or not. As our window size is 2, the first sliding window contains pages *b* and *c*. Since their total new IPs is 4, which is less than 5, we will not crawl their children pages. We then move forward the sliding window. The second sliding window contains pages *c* and *d*, and has 6 new IPs. We thus crawl children pages of both pages *c* and *d*, and get pages *f*, ..., *j*. The last sliding window in this depth contains pages *d* and *e*, which has only 3 new IPs. We thus will not crawl page *e*.

Next, we repeat the above process for each depth of pages until no pages can be selected to crawl. In this example, we will further crawl children pages of pages *i* and *j*, since their total new IPs is 7. Since there are no other windows that can pass the threshold, the crawling will stop after crawling page *m*.

Finally, because the above strategies are only likely but not certain, we may still miss some pages containing new IPs. We thus introduce a random crawling behavior for pages that are originally not selected. Specifically, when a page is not selected by any of our strategy, we still give it a chance to be crawled with a small probability *p*, *e.g.*, 0.05. This random behavior complements our deterministic crawling strategies in a statistical sense.

Implementation: Putting together all the strategies we have discussed so far, we develop the overall Algorithm ADAPTIVE(*T*, *S*, *p*) to realize the LINKSELECTION function for finding Web sites, as Figure 5.8 shows. In particular, lines 3-5 realize the crawling strategy for the target-page pattern. Function

```
Algorithm: ADAPTIVE (T, S, p):
Input: a page pg, IP threshold T, window size S, probability p
Output: a list of selected intra-site links
begin:
1   L = ∅  /* L: the set of selected intra-site links */
2   W = all intra-site links in pg
3   /* deal with target-page pattern */
4   if DEPTH(pg) ≤ 1 then L = L∪ {u| for all u ∈ W}
5   else L = L∪ {u| for all u ∈ W and LINKKEYWORDS(u) = true}
6   /* deal with continues pattern with adaptive crawling*/
7   /* get a set of pages with respect to the window size S */
8   pw = GETPAGEWINDOW(pg, S)
9   E = the set of new IPs found from pages in pw
10  /* check whether there are enough IPs in the window of pages */
11  if |E| ≥ T then L = L∪ {u| for all intra-site links u in pw}
12  /* add random crawling behavior */
13  elif we hit a probability p then L = L∪ {u| for all u ∈ W}
14  return L
end
```

Figure 5.8: Algorithm ADAPTIVE.

DEPTH is to return the depth of a page and function LINKKEYWORDS is to check whether the URL contains some keywords about link pages. Lines 6-11 realize the adaptive crawling strategy for the continuous pattern. Function GETPAGEWINDOW returns a window of pages with the given page *pg* as the last page in the window. Lines 12-13 realize the random crawling behavior.

Local harvest and coverage: Our algorithm, with its parameters, allows us to control the local harvest and coverage. We have three parameters to set: the IP threshold T and the window size S in adaptive crawling, and the random probability p. As our experiments in Section 5.5 show, the combination of these three parameters affect both the local harvest and coverage. It is thus possible to choose the parameters that are likely to have a good balance of local harvest and coverage with respect to user's requirements.

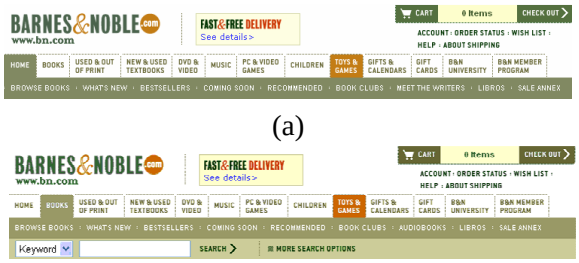
5.4 In-Site Form Searcher

In this section we develop the In-Site Form Searcher for efficiently finding query forms within a site. Section 5.2.1 discussed the simple strategy *Exhaustive(d)*, which can already outperform the base harvest rate with reasonable coverage. However, can we do better? In this section we observe further structure locality (in addition to the "shallow distribution" mentioned in Section 5.2.1) specific to finding query forms. In realizing it, as Section 5.2.3 outlined, we will discuss our observations and discovered patterns for the structural locality of query forms (Section 5.4.1), then formulate strategies and concrete implementations (Section 5.4.2).

5.4.1 Observations and Patterns

Observations: We observe that, as a common feature, most Web sites provide navigational menus to guide users in browsing the sites. Such navigational menus are often presented in order to bring users to important pages, among which of particular interests to us are those containing query forms. To be more concrete, Figure 5.9(a) shows the navigational menu at *http://www.bn.com*. By following the link at the tag `"BOOKS"`, we go to another navigational menu (Figure 5.9(b)) that contains a simple query form and the link `"More Search Options"` to the advanced query form of the book department of *Barnes&Noble*. In fact, we can reach the query forms of all major departments of *Barnes&Noble* by similarly following the links on the tags in Figure 5.9(a). Figure 5.10 illustrates a variety of navigational menus from real-world Web sites.

To verify that the structure locality provides high coverage in finding query forms, we surveyed 100 query forms from the UIUC Web integration repository [17]. These forms are randomly selected from forms that are not on root pages (such root-page forms are always covered, as the In-Site Form Searcher



(a)

(b)

Figure 5.9: Navigational links in *BN.com*.

(a)

(b) (c)

Figure 5.10: More navigational links.

starts with the root page). We find that 87 out of the 100 forms can be reached from the root pages by following navigational links[4].

Reachable Patterns: *First*, as navigational links serve the purpose of connecting from everywhere to the important information of a site, they naturally reach the query forms, which provide a crucial functionality of the site. *Second*, as mentioned in Section 5.2.1, query forms distributed "shallowly" in Web sites, *i.e.*, they are close to the entrances.

5.4.2 Strategy and Implementation

Strategy: There can be hundreds of intra-site links on a Web site. How do we effectively find the navigational links among them? Our method is based on the following two insights.

[4]The remaining 13 forms can mostly be reached by other simple heuristics. For example, most links to "advanced" query forms that could not be directly reached by navigational links are around the simple query forms that can. Here we focus on the navigational links only and do not consider other heuristics.

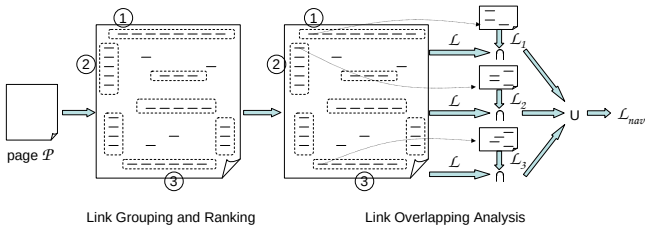


Figure 5.11: NAVMENU: navigational link detection.

First, navigational links in a page are presented with distinguishing visual characteristics, such as alignment, position, size, color, and font. Such characteristics enable the prominent and intuitive visual presentation of navigational links so that users can easily identify them. This observation suggests to analyze the visual pattern of hyperlinks to detect the navigational links. Note that such *visual parsing* approach has also been applied in understanding Web query forms [72].

Second, navigational links are often part of a page "skeleton" of a site that will repeatedly appear in many pages in the site. Such repeating occurrences present the navigational links as shortcuts available everywhere in the site, thus the important information is always reachable. This observation suggests to enforce a *link overlap* analysis across multiple pages.

Implementation: Based on the above insights, we specialize the LINKSELECTION function in Figure 5.5 to an in-site search strategy, Algorithm NAVMENU, for detecting navigational links. Given a page p containing a set of hyperlinks L, it returns a set of candidate navigational links $L_{nav} \subseteq L$. The details of NAVMENU are shown in Figure 5.12. At the high level, the algorithm takes two stages, corresponding to the aforementioned two insights, as illustrated in Figure 5.11.

The first stage, *link grouping and ranking*, explores the visual characteristics for selecting and ranking navigational links. The hyperlinks in L are formed into multiple disjoint *link groups* based on visual parsing. To begin with, a link group is a set of more than, say 3, consecutive links representing a menu, aligned horizontally or vertically. This captures the visual characteristic of most navigational bars in terms of alignment. Further we rank the link groups by other visual characteristics. More specifically,

Algorithm: NAVMENU (p):
Input: a page, p
Output: a list of selected intra-site links, $links$
begin:
1 /* obtain the lines of texts and the URLs by *Lynx* */
2 $(< l_1, ..., l_n >, < u_1, ..., u_m >)$ = lynx(p)
3 /* link grouping*/
4 $t = 0$ /* total number of link groups */
5 **for** each line l_i, $1 \leq i \leq n$ **do**
6 /* NAW: non-anchortext words */
7 **if** line l_i contains single URL u_j and no NAW **then**
8 **if** line l_{i-1} contains multiple URLs or NAW **then**
9 $t = t$+1
10 $g_t = g_t \cup \{u_j\}$
11 /* all the words in l_i are the anchor text of u_j */
12 $wc[g_t]$ = $wc[g_t]$ + number of words in l_i
13 **elif** line l_i contains multiple URLs $\{u_j, ..., u_{j+k}\}$ **then**
14 $t = t$+1
15 $g_t = \{u_j, ..., u_{j+k}\}$
16 /* including both anchortexts and NAW */
16 $wc[g_t]$ = number of words in l_i
17 /* a group should have at least 3 links */
18 remove those groups with $|g_i| \leq 2$
19 /* link group ranking */
20 **for** each group $g_i = \{u_j, ..., u_k\}$, $1 \leq i \leq t$ **do**
21 $total\_size = total\_size + |g_i|$
22 $total\_words = total\_words + wc[g_i]$
23 $total\_dist = total\_dist + m - (j+k)/2$
24 **for** each group $g_i = \{u_j, ..., u_k\}$, $1 \leq i \leq t$ **do**
25 $rank[g_i] = \frac{w_s \times |g_i|}{total\_size} + \frac{w_w \times wc[g_i]}{total\_words} + \frac{w_d \times (m-(j+k)/2)}{total\_dist}$
27 /* link overlap analysis */
28 $ls = \emptyset$
29 **for** each of the top k ranked groups g **do**
30 $ls = ls \cup \{$the first x links in $g\}$
31 $P\_url = < u_1, ..., u_m >$
32 $links = \emptyset$
33 **for** each link $u \in ls$ **do**
34 $child = retrieve(u)$
35 C_url = URLs in $child$
36 $links = links \cup (P\_url \cap C\_url)$
37 **return** $links$
end

Figure 5.12: Algorithm NAVMENU.

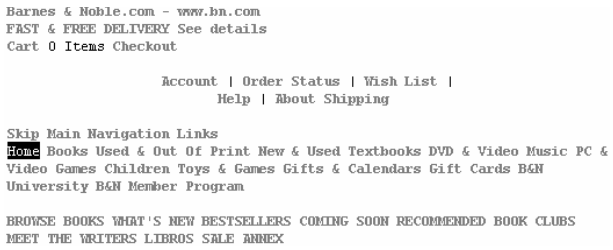


Figure 5.13: The output of Lynx corresponding to Figure 5.9(a).

we take the ranking score of a link group g as the weighted average of three factors. That is, $rank_g = w_s \times size_g + w_w \times wordcount_g + w_d \times dist_g$, where $size_g$ is the number of links in g , $wordcount_g$ is the percentage of anchortext words in all the words positioned in the range of g, and $dist_g$ is the distance between g and the bottom of the page. For example, the top-3 link groups are annotated in Figure 5.11. We note that this formula is simply to capture the important visual characteristics of a link group as a menu, although other heuristics can be developed as well.

The second stage, *link overlap analysis*, identifies the links that repeatedly appear. This captures the observation that the same navigational links are likely to appear in the page itself and the pages referred to by the navigational links. To be more specific, given a navigational link $l \in L$ in p (*e.g.*, Figure 5.9(a)), the target page of l (*e.g.*, Figure 5.9(b)) likely contains l as well. ¿From each of the top k (*e.g.*, $k = 3$ in Figure 5.11) ranked groups, x links (*e.g.*, the first link) are followed. The links in each resulting page (*e.g.*, L_1) are intersected with L, the links in p. The resulting intersections are unioned to form the set of candidate navigational links L_{nav}.

The visual information utilized in NAVMENU can be obtained from the Web page rendering engine of browsers such as IE and Mozilla. To ensure the high efficiency of the crawler, we use the open source browser *Lynx*, which renders a Web page in the text model. For instance, the dumping output of Lynx corresponding to Figure 5.9(a) is shown in Figure 5.13. (The association of the anchortext and the URL of each hyperlink is not shown.) The (anchortexts of) hyperlinks aligned horizontally in a graphical browser will appear in the same line of the textual output. Similarly the hyperlinks aligned vertically

will appear as multiple lines. Note that although *Lynx* only provides approximate visual presentation of a page, compared to graphical rendering engines, and our algorithm even only exploits partial information (such as alignment, size, and distance) from the output of *Lynx*, the result is quite satisfactory according to our experiments in Section 5.5. Further improvements of NAVMENU can be made by exploring more visual characteristics related to navigational links.

Local Harvest and Coverage of Forms: The In-Site Form Searcher captures the reachable pattern of query forms with respect to navigational links, thus can achieve both high harvest and coverage. It only follows navigational links, therefore crawls much less pages than the baseline approach of exhaustively following links, and thus has higher harvest. Navigational links can lead to 87% of query forms (that are not on root pages), therefore the coverage of the form searcher can reach higher than 87% since many forms can be found on root pages.

Moreover, to capture the "shallow" distribution pattern of query forms, we combine NAVMENU with the simple strategy *Exhaustive(d)* in Section 5.2.2 to obtain NAVMENU(d), which follows navigational links within the maximal depth d. The appropriate d for NAVMENU(d) will be empirically determined by the experiments in Section 5.5.

5.5 Experiments

To evaluate the Web Form Crawler, we extensively test each of the core components as well as the entire system, for their (local and global) harvest and coverage. The experimental results verify that 1) by our sampling-then-executing strategy over a small sample of Web sites, we can compare various in-site search strategies and select the appropriate one; and 2) compared to page-based crawling, our best harvest rate is about 10 to 400 times difference, depending on the page traversal schemes used.

To begin with, we have implemented the Web From Crawler, as Figure 5.4 shows, with our control logic built upon several modified open-source softwares. In particular, we build our implementation of the in-site searchers (*i.e.*, the In-Site Site Searcher and the In-Site Form Searcher) based on wget (http://www.gnu.org/software/wget/wget.html). For the In-Site Form Searcher, we revise the text-based browser Lynx (http://lynx.browser.org) to extract visual information of Web pages. We implement the dispatchers in C and use PostgreSQL (http://www.postgresql.org) to support the Site Database and Form Database.

We deployed the crawler, with its parallel architecture, on the HAL PC cluster at the University of Illinois at Urbana-Champaign (http://hal.cs.uiuc.edu). The HAL cluster consists of 100 dual processor machines each with two 500MHz Pentium III Xeon processors, 1 GB of memory and a 9GB SCSI drive. The database servers are Xeon 2.80 GHz dual CPU with 2GB memory.

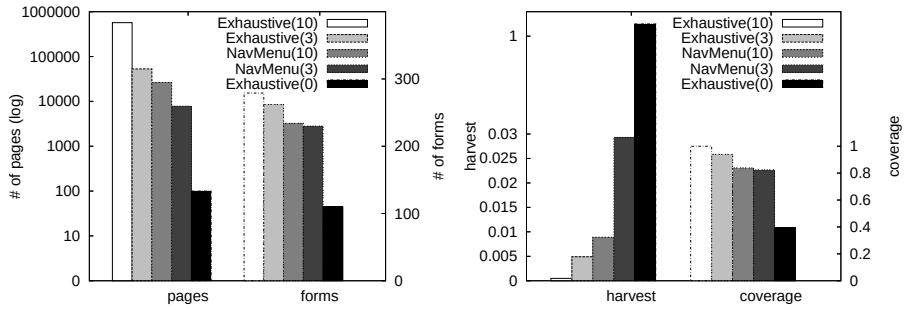
We extensively test the Web Form Crawler in its core components as well as the entire system:

1) Form Finder: We evaluate the local and global performance of the Form Finder in terms of its harvest and coverage.

2) Site Finder: We briefly evaluate the performance of the Site Finder in terms of its harvest and coverage.

3) Overall: We evaluate the Web Form Crawler with a large scale crawling and report the crawling result. We also compare our performance with the one using traditional page-based crawlers.

1a. Local performance of the Form Finder: In this study, we measure the local harvest and coverage of the In-Site Form Searcher under various settings. We randomly choose 100 deep Web sites from the TEL-8 dataset of the UIUC Web Integration Repository [17] as our test set. For each site, we run the In-Site Form Searcher in three cases: Maximal depth d as 0, 3, and 10, denoted as *Navmenu*(0),



(a) Crawling result. **(b)** Performance.

Figure 5.14: Form Finder: Local study.

Navmenu(3), and *Navmenu*(10), respectively. For each case, we measure its local harvest and coverage. As the baseline, we also run the simple strategy of crawling all pages with depth 0, 3 and 10, (*i.e.*, *Exhaustive*(0), *Exhaustive*(3) and *Exhaustive*(10), as Section 5.2.2 introduced). Figure 5.14(a) shows, for each case, the number of pages crawled and the number of forms found. Figure 5.14(b) shows, for each case, the local harvest rate and coverage. As *Exhaustive*(0) is effectively the same as *Navmenu*(0), we only list the result of *Exhaustive*(0).

The result in Figure 5.14 is consistent with our analysis in Section 5.2.2: With a deeper depth, the harvest is lower, while the coverage is higher. Meanwhile, the further structure locality, *i.e.*, navigational menus, developed in Section 5.4 indeed results in better performance. By following navigational menus, we can significantly speed up the harvest while in the meantime maintain a high coverage. In particular, *Navmenu*(3) is the best setting, with a good balance between harvest and coverage.

We note that as the query form classifier (developed in Section 5.2.3) may have false positives, *i.e.*, some non-query forms may be classified as query forms, to have an accurate evaluation of the local harvest and coverage in this small scale study (in contrast to the large scale global study later), we perform a manual inspection to verify query forms and duplicates. The result in Figure 5.14 is the one after manual inspection.

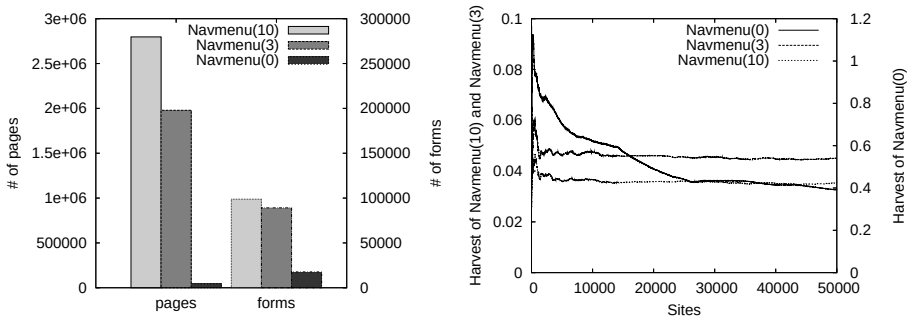
method	*mean harvest*	95% *CI*
Exhaustive(10)	0.114	0.020
Exhaustive(3)	0.119	0.020
Navmenu(10)	0.192	0.028
Navmenu(3)	0.214	0.029
Exhaustive(0)	0.537	0.049

method	*mean coverage*	95% *CI*
Exhaustive(10)	1.0	0.0
Exhaustive(3)	0.898	0.016
Navmenu(10)	0.630	0.026
Navmenu(3)	0.598	0.026
Exhaustive(0)	0.287	0.025

Figure 5.15: Form Finder: Selecting strategy by sampling.

Our manual inspection shows that the more pages we crawl, the more false positives we have in the collected forms. For instance, in depth 0, only 9% forms are false positives, while in depth 10, 60% to 70% forms are. Also, exhaustive crawling, as it crawls more pages, has more false positives than navigational menu crawling. Further, as the 100 deep Web sites we choose in this local study are all "big" sites, the harvest is likely to be underestimated, because for smaller sites, we may not need to crawl many pages to find forms. Therefore, in practice, the global harvest in large scale crawling, where no manual inspection is taken, may increase over 5 times, as our following experiments will show. We believe a more accurate classifier should be and can be developed, but such a topic is beyond the scope of this thesis.

1b. Sampling to select the strategy of the Form Finder: To select a good in-site search strategy for the form finder, we follow the methodology of sampling-then-executing that is mentioned in Section 5.2. In the sampling stage, we apply various strategies over a small sample of 1000 Web sites. For each strategy, we compute its local harvest and coverage over each individual the Web sites. According to the Central Limit Theorem, following the method in Section 5.2, we obtain the 95% confidence interval (CI) for the mean harvest ($\bar{h}$) and mean coverage ($\bar{c}$), respectively, of the underlying population over the whole Web. The results for the five crawling strategies of the form finder are shown in Figure 5.15.

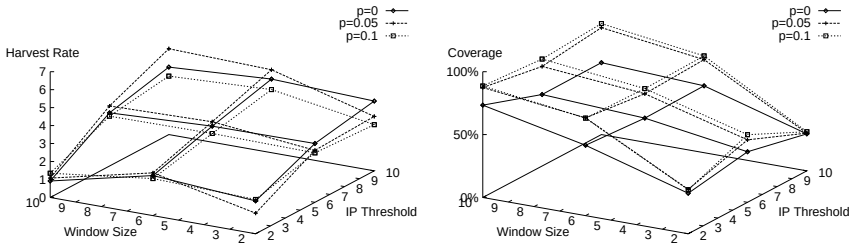


(a) Crawling result. **(b)** Trend of global harvest.

Figure 5.16: Form Finder: Global study.

In the sampling procedure, the following large scale global study and the evaluation of the entire system, we explore automatic query form detection mainly based on the rule-based classifier developed in [22]. However, this automatic detection can result in false positives in both form detection and duplicate removal. The global harvest rate thus will be higher than the one in local study due to the existence of false positives. For instance, one particular error is that the classifier often makes a mistake on considering product configuration forms as query forms, which we specifically removed in the manual inspection. A product configuration form is an HTML form used for configuring features of a specific product, *e.g.*, selecting options of a specific car. For E-commerce sites, it is quite often that each product may have a unique configuration form and thus there are a large number of such forms. Therefore, misclassifying this type of form will result in a significant increase of harvest rate. However, as this issue affects every strategies, we still obtain accurate comparison of the strategies. For example, although the mean harvest obtained from the sampling procedure may not be the same as the real mean harvest across the underlying population, the mean harvests of different strategies still indicate their performance ranking when compared with each other.

1c. Global study of the Form Finder: We then evaluate the global performance of the Form Finder with a large scale crawling. We execute the Form Finder in three cases: Using *Navmenu*(0), *Navmenu*(3), and *Navmenu*(10) as the In-Site Form Searcher respectively. We crawl the same set of 50,000



(a) Harvest. **(b)** Coverage.

Figure 5.17: Site Finder: Local performance.

sites for all cases and compare their performance. Figure 5.16(a) shows, for each case, the number of pages crawled and the number of forms found. Figure 5.16(b) shows, for each case, the trend of global harvest rate. The result shows that, after crawling a few sites, the harvest rate of the Form Finder is quickly stabilized. The harvest of *Navmenu*(0) takes longer to get stable than the other two because it only crawls one page from every site. This result is consistent with our analysis in Section 5.2.2– That is, the structure-driven crawler can indeed maintain stable harvest.

2. Performance of the Site Finder: While we have given detailed analyses of the form finder above, we briefly summarize the results of the site finder, as the main goal of our Web Form Crawler is to collect query forms.

We first evaluate the performance of the Algorithm ADAPTIVE in the In-Site Site Searcher. As Section 5.3 discussed, by tuning the three parameters in ADAPTIVE, *i.e.*, the IP threshold T, the window size S and the random probability p, we should be able to control the local performance of the In-Site Site Searcher for finding site entrances. To verify this argument, we evaluate the In-Site Site Searcher over the *Random100* dataset with an extensive range of combination for $T \in \{2,5,10\}$, $S \in \{2,5,10\}$ and $p \in \{0,0.05,0.1\}$. For each combination, we evaluate its local harvest and coverage. In all the executions, we set the maximal crawling depth d (in Algorithm GENERALINSITESEARCHER) as 10, which is deep enough to cover almost all pages. Figure 5.17 shows the performance.

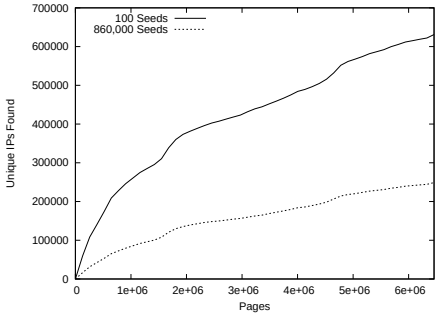
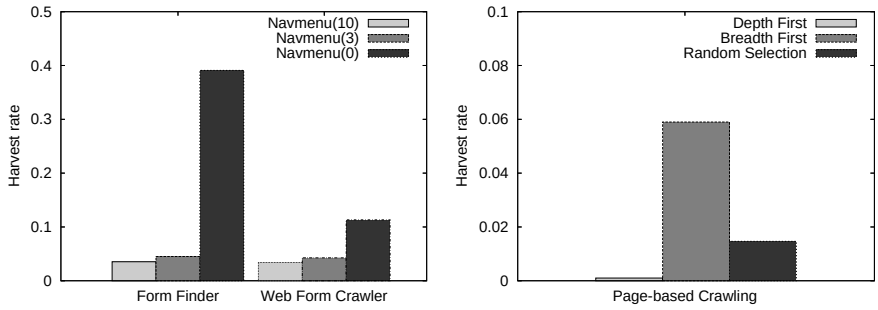


Figure 5.18: Site Finder: Global performance.

From Figure 5.17, we observe interesting trade offs between local harvest and local coverage. In general, when the parameters allow more pages to be crawled (*i.e.*, T is smaller, S is larger, and p is higher), the coverage will be higher, while the harvest will be lower. Overall, any of three parameters can affect the trade off between harvest and coverage. It is thus possible to choose an appropriate parameter setting according to user's desired crawling goal. We apply the sampling method to choose the parameter setting, *i.e.*the specific in-site search strategy, similar to the procedure in Form Finder. For example, the medium values for all the three parameters, *i.e.*, $T = 5$, $S = 5$ and $p = 0.05$, can achieve good harvest as well as reasonable coverage.

With the chosen strategy, we evaluate the performance of the Site Finder by crawling a large set of sites. We execute the Site Finder over the HAL cluster. We test the Site Finder in two cases: Starting from the small *Random100* dataset of 100 IPs and from the large DMOZ list of 860,000 IPs. The top curve in Figure 5.18 shows the global harvest of starting from *Random100* and the bottom one from the DMOZ list. Comparing the two curves, we can see that the more IPs we have in the Site Database, the lower global harvest the Site Finder achieves. However, even with the large starting set of 860,000 IPs, the harvest rate is still reasonably good.

The bottom curve in Figure 5.18 also indicates that the Site Finder can find many new sites that are not indexed by the DMOZ site list. To measure the percentage of IPs we can find beyond the 860,000 IPs from DMOZ, we execute the Site Finder for a long time and collect 2,067,068 IPs, among which



(a) Harvest of the Web Form Crawler.

(b) Harvest of page-based crawling.

Figure 5.19: Evaluation of the entire system.

703,691 overlap with the DMOZ list. That is, only 34% of IPs are indexed by DMOZ and 66% are not. Therefore, it is crucial to develop the In-Site Site Searcher for finding sites besides directly using pre-compiled site lists.

3. Evaluation of the entire system: We next evaluate the overall performance of the Web Form Crawler, including both Site Finder and Form Finder. In particular, the harvest rate of the Web Form Crawler becomes $\frac{\#FormsCollected}{\#PagesCrawledInBothFinders}$. Recall that, in the Site Finder, we have two ways to collect site entrances: Importing from site lists and crawling with In-Site Site Searchers. For the first situation, the Site Finder does not need to crawl pages and thus the harvest rate will be the same as the one in the global study of the Form Finder. For the second situation, as Site Finder also crawls pages, the harvest rate will be lower.

First, overall performance: We can use the result in the global study of the Site Finder to measure the harvest rate in the second situation. Checking our crawling result according to Figure 5.18, we know that starting with 100 sites as seeds, we crawled 110,814 pages to find 50,000 sites. By counting these pages, we can compute the harvest of the Web Form Crawler. Figure 5.19(c) compares the harvest of Web Form Crawler with the corresponding one of Form Finder. In all cases, after counting the pages crawled by the

Site Finder, we can still achieve a good harvest rate, although the harvest is more significantly affected for the In-Site Form Searcher with smaller maximal depth.

Second, comparison to page-based crawling: We compare the harvest of site-based crawling and page-based crawling. As we argued in Section 1, page-based crawling, without focusing on the structure locality, may result in low harvest. We run the traditional page-based crawler to find query forms by simply following links. We test three common link following strategies in page-based crawling: Breadth first, depth first and random selection. For each strategy, we crawl about 50,000 to 150,000 pages and evaluate its harvest.

Figure 5.19(d) shows the result, from which we can see that, site-based crawling achieves better harvest than page-based crawling in finding query forms. For instance, using the depth first strategy, page-based crawling can only find 1 query form in every 1000 pages. Our highest harvest is about 400 times better than this depth first case. Breadth first strategy can achieve better harvest because by following external links and only crawling about 50,000 pages, the page-based crawler is very likely to crawl in the shallow part of many distinct sites and thus behaves similar to a site-based crawler. Even so, our highest harvest is about 10 times better than this breadth first case. Note that while the breadth first strategy "accidentally" explores the structure locality when crawling a relatively small portion of pages, our goal of structure-driven crawling is to formalize and explicitly explore such locality and balance harvest and coverage.

Since the harvest we list here is the one without manual inspection of query forms. We then wonder if the comparison is still valid. Our answer is yes. Recall that our manual inspection shows that the more pages we crawl, the more percent of false positives we have in the collected forms. Compared to site-based crawling, a page-based crawler will be more likely to touch the large number of pages in

deeper depth and thus have more false positives. Therefore, our comparison here in fact disfavors the site-based crawling, although the result of site-based crawling is still better.

We notice that the initial harvest of page-based crawling in Figure 5.19(d) is higher than our average estimation of 6.6×10^{-5}. There are three reasons: First, since many query forms are duplicated in a large number of pages in their own sites, *e.g.*, the keyword book search in *BN.com* may appear in many pages, the initial chance to see a query form in page-based crawling is thus higher. When we crawl more and more pages, the harvest of page-based crawling will slow down and become worse and worse, since many query forms are already seen. Second, the false positive problem in the query form classifier also makes the harvest significantly higher than it real value. Third, our survey of the scale of query forms are done in April 2004. With the rapid growth of the deep Web, we believe there are more query forms available on the Web now.

5.6 Conclusion

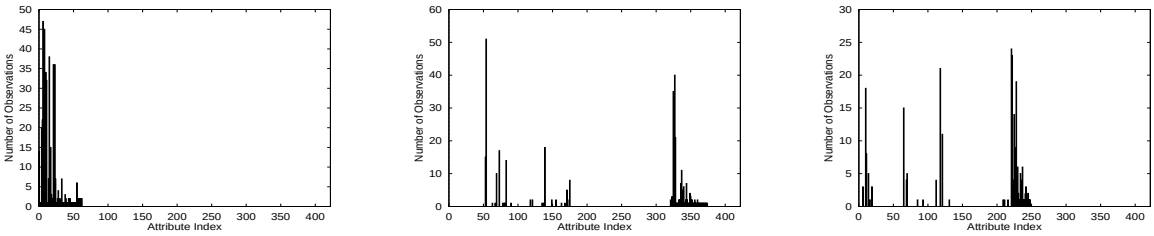
This chapter aims at building a crawler for collecting query forms on the Web. Although critical to information search and integration over the deep Web, such a problem has not been extensively studied. As a new attempt, we abstract this problem as object-focused, topic-neutral crawling and propose a structure-driven crawling framework for such a crawling task by observing the existence of structure locality of query forms. We develop the Web Form Crawler to realize the framework. The experimental results show that our crawler can maintain stable yields in the entire crawling process and thus we can pursue a yield-guided crawler design. Such features are not supported by existing focused crawlers. Compared to page-based crawling, our best harvest rate is about 10 to 400 times difference, depending on the page traversal schemes used.

Next, we will discuss our work on clustering deep Web sources into their domain hierarchy in Chapter 6, which is the second requisite task to enable automatic large scale matching.

Chapter 6

Clustering Query Schemas into a Domain Hierarchy

Since our holistic matching algorithms require the input schemas from the same domain, given a set of collected schemas across various domains (from Chapter 5), we need to cluster these schemas into their domain hierarchy. To distinguish schemas in query interfaces with traditional schemas, we name the former *query schemas*, which contain a set of attributes in their query interfaces, *e.g.*, {author, ..., publisher} for *amazon.com*; {city, ..., rent} for *apartments.com*. Our observations show that query schemas are right "representatives" for structured sources: First, they are readily *available*, on the "surface" of online databases, and thus can be easily "crawled." Second, they are *discriminative*: The query schema characterizes its *object domain* (*e.g.*, Books, Movies) by its *query capabilities* (*e.g.*, author, director). Such observations motivate us to propose model-differentiation as a new objective function for clustering, which allows principled statistical measure for determining cluster homogeneity.



(a) Airfares. **(b)** Movies. **(c)** Hotels.

Figure 6.1: Attribute frequencies of different domains.

6.1 Motivation

Our clustering approach is motivated by our observations on the deep Web. In particular, to better understand the characteristics of schemas in different domains, we again explore the TEL-8 dataset in the UIUC Web Integration Repository [17]. We have two observations pertinent to our focus of schema clustering, which we will report in this section.

First, we observe that query schemas are *discriminative* representatives of structured sources. Specifically, we count attribute frequencies for each domain (*i.e.*, the aggregate occurrences of an attribute across all sources in the same domain). Figure 6.1 lists the attribute frequencies (y-axis) of 3 domains (Airfares, Hotels and Movies) over all the attributes (x-axis) in the 8 domains. We observe that each domain contains a dominant range of attributes, distinctive from other domains. For example, Airfares only covers the first 53 attributes and does not overlap with Movies. Hotels has its dominant range of attributes from index 200 to 250 (while overlapping with Airfares in some of the first 53 attributes).

Further, some attributes are only observed in one domain– These *anchor attributes* make their domains more distinguishable. For instance, make and model are anchor attributes for Automobiles, and ISBN for Books. We observed that most schemas indeed contain anchor attributes. In particular, our dataset indicates that 457 out of the total 494 interfaces, accounting for 92.5%, contain some anchor attributes. The prevalence of anchor attributes motivates our bootstrapping techniques (Section 6.3.1).

We believe that the existence of anchor attributes might not be a unique phenomenon for schema data– For other types of transactional data, it is likely that a cluster will contain some *anchor items* that are characteristics of the cluster.

Second, we observe that the aggregate schema vocabulary of sources in the same domain tends to converge at a relatively small size with respect to the growth of sources. Figure 6.2 shows, for each domain, the growth of vocabularies as sources increase in numbers. The curves clearly indicate the convergence of vocabularies. Since the vocabulary growth rates (*i.e.*, the slopes of these curves) decrease rapidly, as sources proliferate, their vocabularies will tend to stabilize. This observation indicates that homogeneous sources (in the same domain) share some *concerted* vocabulary of attributes. Note that we also exploit this observation for the task of schema matching in the MGS framework in Chapter 2.

These two observations together motivate our approach: The first "discriminative" observation suggests using query schemas as "representatives" of sources in the source organization, which is essentially a clustering problem. Our goal is thus to cluster structured Web source into their domain hierarchy. By viewing a schema as a transaction and thus a special type of categorical data, we abstract our problem of source organization as the clustering of categorical data. Further, the second "concerted" observation leads us to hypothesize the existence of a hidden schema model (for each domain), which generates the observed query schemas. We thus pursue model-based clustering (Section 6.2). Finally, the "discriminative" observation further hints a novel objective function, model-differentiation, which seeks to maximize statistical heterogeneity among models in clustering.

6.2 MD-Based Clustering

As just abstracted and motivated, we are pursuing a MD-based approach to cluster query schemas. In the literature, model-based clustering has been widely discussed. The general idea can be stated as:

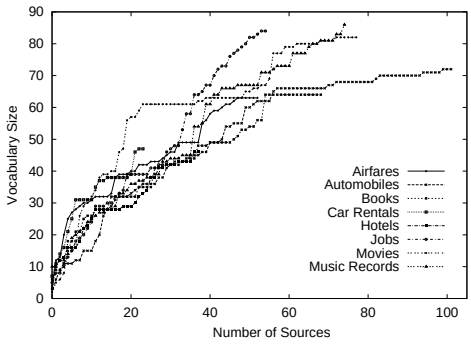
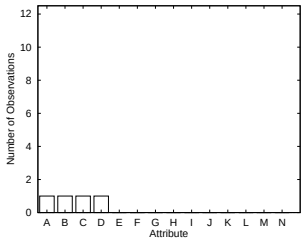


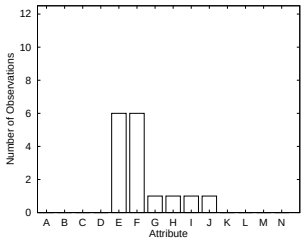
Figure 6.2: Schema vocabularies.

The population of interest consists of G clusters, generated by G different models. Given a set of data points (a set of schemas) $\mathbf{X} = \{\mathbf{x_1}, ..., \mathbf{x_n}\}$, where each $\mathbf{x_i}$ is independently generated from one of the G models, $\mathcal{M}_1$,...,$\mathcal{M}_G$, the probability of generating $\mathbf{x_i}$ in the kth model is $Pr(\mathbf{x_i}|\mathcal{M}_k)$. A clustering of $\mathbf{X}$ is a partition of $\mathbf{X}$ into G groups: denoted by $(\mathbf{X};P) = (C_1, ..., C_G)$, where P partitions $\mathbf{X}$. The objective of model-based clustering is to identify the partition P that all $\mathbf{x_i}$ generated from the same model $Pr(\bullet|\mathcal{M}_k)$ are partitioned into a single group.

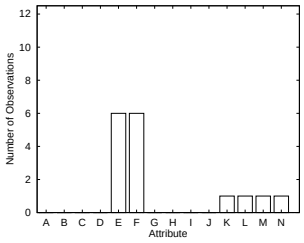
To realize this general model-based clustering of query schemas, we design a model as a multinomial distribution (Section 6.2.1) and develop model-differentiation as the new *objective function* of clustering based on statistical hypothesis testing. Specifically, guided by this objective function, we adopt the commonly used χ^2 testing. (Section 6.2.2). Unlike the clustering work in statistic software, which also use χ^2 testing, we apply it for categorical data based on the generative model. Since we are pursuing a hierarchical clustering approach, we apply the widely used HAC (hierarchial agglomerative clustering) algorithm, which needs a measure to quantify the "similarity" between two clusters. In particular, we derive a new similarity measure from the MD objective function (Section 6.2.3).



(a1) $C_{<1,2>}$.

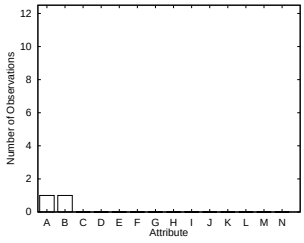


(a2) C_3.

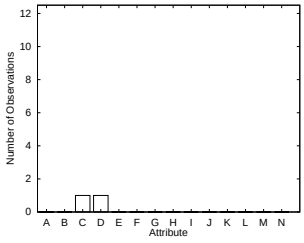


(a3) C_4.

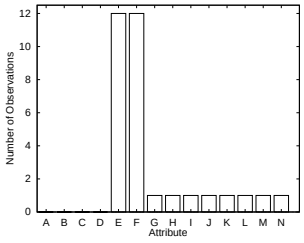
(a) Multinomial distributions of the clustering result by combining C_1 and C_2.



(b1) C_1.



(b2) C_2.



(b3) $C_{<3,4>}$.

(b) Multinomial distributions of the clustering result by combining C_3 and C_4.

Figure 6.3: Comparison of two possible clustering results.

6.2.1 Hypothesis Modeling

To develop the MD-based clustering, we need to define the generative model. To begin with, we first introduce our model definition as multinomial distribution. Specifically, we assume attributes are independent each other, which is a commonly used assumption for text data [58]. Then we describe how a model generates a schema in a statistical way and further how to generate a cluster of schemas.

First, to define the model for the task of schema clustering, we need to describe what is a schema. We view a query schema as a set of attributes for a query interface, as we abstracted in the MGS and DCM frameworks. For instance, for *amazon.com*, the query schema Q_{az} is $\{\mathsf{author}, \ldots, \mathsf{publisher}\}$. For simplicity, in later examples, we denote attributes in letters A, B,....

Our first attempt is to consider a schema as a set of *distinct* attributes. Therefore, we view the generation of a schema as *sampling without replacement* [14] from a set of attributes, which means the result of a trial (to select an attribute) is not the same as any previous trials. (The trials are therefore

"stateful".) That is, we can consider a schema with n attributes as an experiment with n trials; once one attribute is selected, it will not be selected again in the subsequent trials. However, while this model is accurate, its "stateful" trials result in complicated homogeneity testing.

Our second attempt is to approximate the generation process by *sampling with replacement* [14], where the attributes can be repeatedly selected in a schema. With this alternative strategy, to generate a schema Q in some cluster C, the model $\mathcal{M}$ behind C is a *multinomial model* with parameters $p_1,...,p_N$. More specifically, a multinomial model $\mathcal{M}$ for C consists of an exhaustive set of N mutually exclusive events (In our problem, the events are in fact the attributes.) $A_1,...,A_N$ (which covers all the attributes observed in C) with associated probabilities $p_1,...,p_N$, $\sum_{j=1}^{N} p_j = 1$. We denote $\mathcal{M}$ as $\mathcal{M} = \{A_1{:}p_1,...,A_N{:}p_N\}$. Each trial of $\mathcal{M}$ generates one of the N events. The probability of generating an attribute A from $\mathcal{M}$ in a single trial is

$$Pr(\mathsf{A}|\mathcal{M}) = \begin{cases} p_i, & \exists i : \mathsf{A} = A_i \\ 0, & otherwise \end{cases} \tag{6.1}$$

Next, we discuss the generation of a schema in cluster C from $\mathcal{M}$. Under this multinomial model, a schema Q is characterized by its observed attributes (and their frequencies). We thus view Q (of length n) as $Q = \{A_1{:}y_1,...,A_N{:}y_k\}$, $\sum_{i=1}^{N} y_i = n$, where y_i is the frequency (number of occurrences) of attribute A_i in Q. For a schema with distinct attributes, y_i is either 0 or 1. For instance, for the query schema Q_{az} of *amazon.com*, the frequency of author, y_{author}, is 1. (We discuss later that this model can generate schemas with duplicate attributes.) That is, by definition of standard multinoimal distribution [14], Q (of length n) is generated from $\mathcal{M}$ as the result of n independent (therefore "stateless") trials with the following probability:

$$Pr(Q|\mathcal{M},n) = n!\prod_{i=1}^{N}\frac{Pr(A_i|\mathcal{M})^{y_i}}{y_i!}. \tag{6.2}$$

Example 26: Consider a cluster C with 4 schemas: Q_1:{A,B,C}, Q_2:{A,B}, Q_3:{C,D}, and Q_4:{C,D,E}. The model $\mathcal{M}$ contains 5 attributes (events): A, B, C, D and E, with probabilities p_1, p_2, p_3, p_4 and p_5 respectively. Under our multinomial modeling, we view a schema as a set of attribute frequencies (*i.e.*, y_{ji}). For example, Q_1 = {A:1, B:1, C:1, D:0, E:0}. In particular, $y_{11} = 1$ since A occurs once in Q_1. The probability of generating Q_1 is $Pr(Q_1|\mathcal{M},3) = 6p_1p_2p_3$.

Then, we discuss how we statistically view a cluster of schemas. Consider a cluster of schemas $C = \{Q_1, Q_2, ..., Q_m\}$, where each schema Q_j (with length n_j) is generated by the same model $\mathcal{M}$ = $\{A_1{:}p_1,...,A_N{:}p_N\}$. Since each Q_j is a multinomial experiment of n_j trials, we can view C as an experiment with $\sum_{j=1}^{m} n_j$ trials by concatenating the trials in all schemas. That is, we consider that C is a series of sampling from the same multinomial distribution $\mathcal{M}$ (*i.e.*, the same $p_1,...,p_N$), with all these independent trials. The theoretical explanation is as follows: Let all Q_j = $\{A_1{:}\mathbf{y_{j1}},...,A_N{:}\mathbf{y_{jN}}\}$, where $\mathbf{y_{ji}}$'s are random variables denoting the frequencies of A_i, share the same multinomial distribution $\mathcal{M}$ = $\{A_1{:}p_1,...,A_N{:}p_N\}$. For the entire C, we define new random variables $\mathbf{z_1},...,\mathbf{z_N}$ as aggregate attribute frequencies. That is, $\mathbf{z_i} = \sum_{j=1}^{m} \mathbf{y_{ji}}$. In our extended report [40], we show that $\mathbf{z_1}, ..., \mathbf{z_N}$ are also sampled from the same multinomial distribution $\mathcal{M}$ with $\sum_{j=1}^{m} n_j$ trials. Therefore, under this multinomial view, we can express C as aggregate attribute frequencies, *i.e.*, C = $\{A_1{:}z_1,...,A_N{:}z_N\}$.

Example 27: Continue on the cluster C in Example 26, by considering C is generated by a multinomial distribution and computing z_i as $\sum_{j=1}^{m} y_{ji}$, we can express cluster C as {A:2, B:2, C:3, D:2, E:1}. For example, A has frequencies 2 because it occurs once in both Q_1 and Q_2.

	A_1	A_2	A_3	...	A_n	*sum*
C_1	O_{11}	O_{12}	O_{13}	...	O_{1n}	X_1
C_2	O_{21}	O_{22}	O_{23}	...	O_{2n}	X_2
...	...	...	...	...	...	...
C_m	O_{m1}	O_{m2}	O_{m3}	...	O_{mn}	X_m
sum	Y_1	Y_2	Y_3	...	Y_n	S

Figure 6.4: Contingency table for testing.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	*sum*
$C_{<1,2>}$	1	1	1	1	0	0	0	0	0	0	0	0	0	0	4
C_3	0	0	0	0	6	6	1	1	1	1	0	0	0	0	16
C_4	0	0	0	0	6	6	0	0	0	0	1	1	1	1	16
sum	1	1	1	1	12	12	1	1	1	1	1	1	1	1	36

Figure 6.5: Example of model-differentiation testing.

The simple multinomial modeling simplifies hypothesis homogeneity testing (by directly fitting the contingency table as shown in Section 6.2.2). However, the modeling is inaccurate: It may generate some schemas that are not observable in the real world. For instance, it may generate a schema $\{$author, author, title$\}$, where author is repeated twice. While the modeling seems crude (like other typical "independent" assumptions in, say, Naive Bayes Classifier for text), our empirical study shows that the simple model performs well.

As a remark, this modeling is much simpler than what we define in our previous work MGS [37]. The MGS work addresses matching schemas across sources in the same domain. (Therefore, the work of this chapter is a preliminary step to provide input for MGS.) The MGS modeling assumes a two-level model structure to capture concepts and synonyms for the goal of synonym discovery. This work assumes a much simpler model because it is sufficient to capture the attribute frequencies across different domains for the purpose of clustering.

In this section, we develop the generative model. Next, we introduce the new objective function, model-differentiation, for clustering schema data and present the χ^2 testing to realize the MD function.

6.2.2 Model-Differentiation: A New Objective Function

Clustering must be guided by some *objective function* that specifies the property of the ideal clusters. Regardless of the objective function, the basic idea of clustering is to put similar data together and dissimilar data apart. For model-based clustering, similar data might be generated from the same underlying model, while dissimilar data from different models. Thus, we achieve better clustering result when the underlying models are more distinguishable.

Example 28: As a running example, assume we are given four clusters of schemas, referred to as dataset I: C_1:{A:1, B:1}, C_2:{C:1, D:1}, C_3:{E:6, F:6, G:1, H:1, I:1, J:1} and C_4:{E:6, F:6, K:1, L:1, M:1, N:1}. Now assume we want to generate 3 clusters (G = 3). To reduce the number of clusters to three, we need to combine two clusters into one. We denote the combination of clusters C_k and C_l as $C_{<k,l>}$.

We compare two possible clustering results, as illustrated in Figure 6.3. The first result (Figure 6.3(a)) combines C_1 and C_2, while the second result (Figure 6.3(b)) combines C_3 and C_4. Figure 6.3(a) is not as good as Figure 6.3(b) because the distributions of C_3 (Figure 6.3(a2)) and C_4 (Figure 6.3(a3)) are similar (and hence the schemas generated from these two models will also be similar). Figure 6.3(b) is better because the attributes with non-zero frequencies in the three clusters do not overlap.

Therefore, we define the objective function of clustering as some function $\mathcal{H}$ that characterizes the heterogeneity of models under a partition P, denoted by $\mathcal{H}(\mathbf{X};P)$. The goal of clustering is to find the partition P that maximizes function $\mathcal{H}$, *i.e.*, $\arg\max_P \mathcal{H}(\mathbf{X};P)$. In statistics, the homogeneity of distributions can be measured by *test of homogeneity* using statistical hypothesis testing. More specifically, if we have a partition function P partitioning $\mathbf{X}$ into clusters $C_k(1 \leq k \leq G)$, we can test the hypothesis "$C_k(1 \leq k \leq G)$ are sampled from the same distribution" with standard testing approaches. The

result of testing is a probabilistic variable λ to indicate the confidence that we accept the hypothesis that those distributions are the same. Thus the heterogeneity of models is $1-\lambda$. Formally, the MD-based clustering is to find

$$\begin{aligned}\arg\max_P \mathcal{H}(\mathbf{X};P) &= \arg\max_P \mathcal{H}(C_1,...,C_G) \\ &= \arg\max_P \{1-\lambda(C_1,...,C_G)\} \\ &= \arg\min_P \lambda(C_1,...,C_G), \end{aligned} \tag{6.3}$$

where $\lambda(C_1,...,C_G)$ is the result of hypothesis testing on a partition P with G clusters.

More specifically, given a partition P on the observed data $\mathbf{X}$, we apply χ^2 hypothesis testing to compute $\lambda(C_1,...,C_G)$. In statistics, χ^2 testing can be used to test the homogeneity among multiple clusters with multinomial distributions by constructing a *contingency table*. Since we show that a cluster of schemas is also generated by a multinomial distribution, we can directly apply the test of homogeneity by fitting the attribute frequencies in the cluster into the contingency table, which reflects the fact that our modeling simplifies the testing. For arbitrary models, it deserves further research efforts to figure out how to fit them into the contingency table.

Formally, assume there are m clusters $C_1,...,C_m$, and each of them is generated from its own multinomial distribution (as defined in Section 6.2.1). There are n different events (attributes) altogether, denoted by $A_1,...,A_n$. Figure 6.4 is the contingency table to show this set of data. In particular, O_{ij} stands for the attribute frequency of A_j in cluster C_i. X_i is the sum of all the O_{ij} in ith row and Y_j is the sum of all the O_{ij} in jth column. That is, $X_i = \sum_{j=1}^{n} O_{ij}$ and $Y_j = \sum_{i=1}^{m} O_{ij}$. S is the sum of all O_{ij} in the table. Thus $S = \sum_{i=1}^{m} X_i = \sum_{j=1}^{n} Y_j$.

We want to test the hypothesis: $\forall j, 1 \leq j \leq n$, $p_{j1} = p_{j2} = ... = p_{jm} = \frac{Y_j}{S}$, where p_{ji} is the probability of observing attribute A_j in cluster C_i. This hypothesis is tested by considering the random variable

$$D^2(C_1,...,C_m) = \sum_{i=1}^{m}\sum_{j=1}^{n}[\frac{(O_{ij} - X_i \times \frac{Y_j}{S})^2}{X_i \times \frac{Y_j}{S}}]. \tag{6.4}$$

It can be shown that D^2 has asymptotically a χ^2 distribution with $(n-1)(m-1)$ degree of freedom, denoted by df [2].

We have to use both D^2 and df to decide how similar the m clusters are. D^2 value itself is not a valid indicator for the similarity of clusters without being qualified the degree of freedom. Therefore we need to translate these two values into a single similarity measure. In statistics, we can compute the P-value given D^2 and df, denoted by $PV(D^2, df)$. The P-value is the probability value λ in Equation 6.3, indicating the confidence that we accept the hypothesis that the m clusters are generated from the same distribution. The objective function $\mathcal{H}$ is then

$$\mathcal{H}(C_1,...,C_G) = 1 - PV(D^2, df). \tag{6.5}$$

The computation of P-value is expensive and requires numerical integration. Therefore, in practice, we develop an alternative measure, $\tilde{\mathcal{H}}$, by applying a normalized D^2 value. In particular, to make the D^2 values of different degrees of freedom (resulted from different clusters) comparable, we use the D^2 values with a commonly adopted significance level 0.5% as the normalization factors, denoted by $D_s^2(df)$, with different degrees of freedom. We consider $\tilde{\mathcal{H}}$ the ratio between the computed D^2 value and the D_s^2 with the same df:

$$\tilde{\mathcal{H}}(C_1,...,C_G) = \frac{D^2}{D_s^2(df)}. \tag{6.6}$$

Algorithm: GE_{hac}:
Input: SchemaSet **X**, ObjectiveFunction $\mathcal{F}$, NumberOfClusters G
Output: G clusters
begin:
1 **/* Form a list of initial V clusters */**
2 $C_k = X_k$, $(1 \leq k \leq V)$
3 **/* Derive similarity measure */**
4 s = a similarity measure derived from $\mathcal{F}$
5 **/* HAC main framework */**
6 **for** $K = V, V-1, ..., G$
7 **/* Compute pairwise similarities */**
8 $k^*, l^* = \arg\min_{k,l} s(C_k, C_l)$, $(1 \leq k < l \leq K)$
9 **/* Merge the most similar two clusters*/**
10 $C_{<k^*,l^*>} = \text{MERGE}(C_{k^*}, C_{l^*})$
end

Figure 6.6: General HAC algorithm GE_{hac}.

Example 29: Consider the first clustering result in Example 28, we want to test the hypothesis that these three clusters are generated from the same distribution. The corresponding contingency table of this scenario is listed in Figure 6.5. Applying Equation 6.4, we get $D^2(C_{<1,2>}, C_3, C_4) = 34.33$ and $df(C_{<1,2>}, C_3, C_4) = (14-1) \times (2-1) = 13$. the D_s^2 value for 0.5% with $df = 13$ is 29.82. By applying Equation 6.6, $\mathcal{H}(C_{<1,2>}, C_3, C_4) = \frac{34.33}{29.82} = 1.15$.

Consider the second clustering result in Example 28 similarly, we get $D^2(C_1, C_2, C_{<3,4>}) = 65.67$ and $df(C_1, C_2, C_{<3,4>}) = (14-1) \times (2-1) = 13$. We then have $\mathcal{H}(C_1, C_2, C_{<3,4>}) = \frac{65.67}{29.82} = 2.2 > \mathcal{H}(C_{<1,2>}, C_3, C_4)$, which means the second clustering result is better than the first one.

6.2.3 General HAC Algorithm and MD-Based Similarity Measure

For constructing the domain hierarchy as motivated in Section 6.1, we adopt the general HAC clustering approach, which is widely used for data clustering [43]. Figure 6.6 illustrates the general HAC framework [51]. In HAC, we need to measure the similarity of clusters. That is, given a set of clusters, $C_1, ..., C_V$, we compute all the pairwise values $s(k, l)$, where s is a similarity function from the objective

function of clustering. The criterion of defining similarity function $s(k,l)$ is to maximize the objective function in each step. The two clusters with the smallest $s(k,l)$ are merged in each iteration. The algorithm stops when there are G clusters left.

Specifically, for our MD-based clustering, we derive $s(k,l)$ from $\mathcal{H}(\mathbf{X};P)$ (defined in Section 6.2.2) as follows: In each iteration of HAC, we merge the clusters with the smallest $\mathcal{H}$ value (*i.e.*, the most similar two models) and therefore we define $s(k,l)$ to be

$$s(k,l) = \mathcal{H}(C_k, C_l). \tag{6.7}$$

Example 30: Consider the dataset I in Example 28 as the input of GE_{hac}. Assume we want to generate 3 clusters. We compute all the pairwise similarities with Equation 6.7 and get $s(1,2) = 0.43$, $s(1,3) = 0.96$, $s(1,4) = 0.96$, $s(2,3) = 0.96$, $s(2,3) = 0.96$, and $s(3,4) = 0.04$. It is clearly to see that C_3 and C_4 are most similar. Thus the clustering result is C_1, C_2 and $C_{<3,4>}$.

6.3 Clustering Query Schemas: Algorithm MD_{hac}

In this section, we present the concrete algorithm MD_{hac} (denoting MD-based HAC algorithm) by solving the difficulty of applying the MD-based clustering. To test the heterogeneity of models with hypothesis testing (Section 6.2.2), we have to face one challenge: When the observations of events are not sufficiently large, the value of D^2 may not be closely converged to χ^2 distribution and thus affects the value of $\mathcal{H}$. In particular, the χ^2 test requires each event (attribute in our case) has at least 5 observations to ensure the approximation of χ^2 distribution [2]. However, the input data are initially collected without being grouped and thus cannot satisfy this requirement.

Algorithm: MD_{hac}:
Input: SchemaSet **X**, NumberOfClusters *G*
Output: G clusters
begin:
1 **/* Form the initial clusters */**
2 $\mathcal{C}$ = DATAGROUPING(**X**)
3 **/* Move loner interfaces into $\mathcal{N}$ */**
4 $\mathcal{C}$, $\mathcal{N}$ = GROUPSELECTION($\mathcal{C}$)
5 **/* Standard HAC clustering with new measure */**
6 $\mathcal{C}$ = CLUSTERINGHAC(G, $\mathcal{C}$)
7 **/* Classify loners into accomplished clusters */**
8 $\mathcal{C}$ = LONERHANDLING($\mathcal{N}$, $\mathcal{C}$)
9 **/* Build the domain hierarchy with HAC approach */**
10 BUILDHIERARCHY($\mathcal{C}$)
end

Figure 6.7: Algorithm MD_{hac}.

To address this problem of insufficient observations, we design pre-clustering and post-classification techniques. Pre-clustering is to pre-cluster the data into groups with sufficient observations to satisfy the requirement of hypothesis testing. Post-classification is to classify the insufficient *loner* schemas excluded by bootstrapping into the accomplished clusters.

In our development, pre-clustering consists of two steps: data grouping and group selection. Data grouping is to merge the data into groups by using deterministic rules. After grouping, some groups contain sufficient observations, while others not. Group selection only selects those groups with sufficient observations to participate in the HAC clustering. We consider the insufficiently observed schemas as loner schemas. Post-classification is essentially the classification of loners into the completed clusters, which we call loner handling in our implementation.

Figure 6.7 shows Algorithm MD_{hac}: First, DATAGROUPING pre-clusters data into groups based on the corollaries developed from the existence of anchor attributes (Section 6.3.1). Second, GROUPSELECTION excludes the loner schemas with loner threshold N (Section 6.3.2). Third, CLUSTERINGHAC clusters the remaining groups with the standard HAC algorithm and Equation 6.7 as the similarity mea-

sure. Fourth, LONERHANDLING classifies the loner schemas into the accomplished G clusters (Section 6.3.3). Finally, BUILDHIERARCHY again applies the HAC algorithm to build the hierarchical tree of domains (by considering each cluster as one domain).

6.3.1 Data Grouping

Our pre-clustering technique leverages the existence of *anchor attributes* to group schemas deterministically. Our exploration for the schemas of the 8 domains indicates that most schemas contain anchor attributes (Section 3.1). Specifically, an anchor attribute is essentially an attribute with non-zero probability only for one cluster. More formally,

Definition 3: Given a clustering partition $C_1, ..., C_G$ and assume the model under C_k is $\mathcal{M}_k$, an attribute A is an *anchor attribute* if there is only one C_k that contains A, *i.e.*, $Pr(\mathsf{A}|\mathcal{M}_k) > 0$ and $Pr(\mathsf{A}|\mathcal{M}_l) = 0$ for $l \neq k$. A schema is a *distinguishable schema* if it contains at least one anchor attribute.

Definition 3 implies the following corollaries:

Corollary 1: If A is an anchor attribute with $Pr(\mathsf{A}|\mathcal{M}_k) > 0$ and A is observed in a schema Q with length n, then $Q \in C_k$, $Pr(Q|\mathcal{M}_k, n) > 0$ and $Pr(Q|\mathcal{M}_l, n) = 0$ for any $l \neq k$.

Proof Assume Q = $\{A_1{:}y_1,...,A_s{:}y_s\}$ of length n and $\mathsf{A} = A_t$, $y_t > 0$ since A is observed in Q. By applying Equation 6.2, we have $Pr(Q|\mathcal{M}_l, n) = n! \prod_{i=1}^{s} \frac{Pr(A_i|\mathcal{M}_l)^{y_i}}{y_i!}$. For $l \neq k$, $Pr(\mathsf{A}|M_l) = 0$ according to definition of anchor attribute, thus we have $Pr(Q|\mathcal{M}_l, n) = 0$. Since Q must belong to some cluster, Q has to be clustered into C_k, thus $Q \in C_k$ and $Pr(Q|\mathcal{M}_k, n) > 0$.

Corollary 2: If Q_1 is a distinguishable schema and $Q_1 \subseteq Q_2$, Q_2 is also a distinguishable schema and belongs to the same cluster as Q_1.

Proof Assume $Q_1 \in C_k$. Q_2 must belong to some cluster C_l. If $l \neq k$, Q_1 becomes a schema containing overlapping attributes of C_k and C_l. Thus $Pr(Q_1|\mathcal{M}_l, n) > 0$, which contradicts the assumption that Q_1 is a distinguishable schema. Therefore we have $l = k$, which means Q_2 is in the same cluster as Q_1.

Corollary 2 indicates that if all the schemas are distinguishable schemas, the containment relation is correct in grouping data. Guided by Corollary 2, we group the query schemas by putting all the schemas satisfying Corollary 2 into one cluster. More specifically, we first randomly select a schema Q, and put all the query schemas Q_i satisfying $Q \subseteq Q_i$ or $Q_i \subseteq Q$ into the same bucket of Q. We then evaluate all the Q_i just added recursively until no satisfied schema can be found. It can be shown that the output of data grouping is not affected by the random selection of schemas.

However, Corollary 2 requires that the schemas are distinguishable schemas. Since it is difficult to affirm whether a schema is distinguishable before clustering, we design a heuristic by observing the difference of the *containing set* of distinguishable and indistinguishable schemas. We define the containing set of a schema Q, denoted by $S(Q)$, as all the Q_is satisfying $Q \subseteq Q_i$ in the dataset. Intuitively, for a distinguishable schema Q, the schemas in $S(Q)$ are in one domain (based on Corollay 2) and hence they should be more overlapping in attributes; While for an indistinguishable schema Q, the schemas in $S(Q)$ come from multiple domains and they should be more different in attributes. Hence, we design a step of *schema type checking* before grouping: For each schema Q, we compute its containing set $S(Q)$. Then for any Q_i and Q_j in $S(Q)$, we compute their distance $d(i, j)$ as $\frac{|Q_i \cap Q_j|}{|Q_i \cup Q_j|}$. If there exists $d(i, j) < \theta$, where θ is a threshold value, we consider Q an indistinguishable schema and exclude it to participate in data grouping. (In fact, the excluded schemas effectively become loner schemas in group selection). In our experiment, we set $\theta = 0.2$. We assume the remaining schemas are all distinguishable schemas and apply Corollary 2 to group them.

Example 31: Consider a set of 8 schemas: Q_1:{C}, Q_2:{A,B}, Q_3:{A,B,C,E}, Q_4:{A,D}, Q_5:{A,B,D,E}, Q_6:{C,F}, Q_7:{C,F,G}, and Q_8:{C,H}. First, we do the schema type checking on every schema. In particular, Q_1's containing set $S(Q_1)$=$\{Q_3,Q_6,Q_7,Q_8\}$. Computing the pairwise distance of schemas in $S(Q_1)$, we know the minimal distance is $d(3,7) = 1/6 < 0.2$. Therefore, Q_1 is indistinguishable schema and excluded for grouping. Similarly, we check other schemas and they all pass this checking.

Next, we start to group the remaining schemas by randomly choosing a schema, say Q_2. Then we find $Q_2 \subseteq Q_3$ and $Q_2 \subseteq Q_5$. By recursively evaluating Q_3 and Q_5, we find $Q_4 \subseteq Q_5$ and no more schemas can be incorporated. Therefore Q_2, Q_3, Q_4 and Q_5 are in one group. We repeat this process on the remaining schemas and find Q_6 and Q_7 are in another group and Q_8 itself is in the third group. The excluded Q_1 is considered as an individual group. Hence, data grouping outputs four groups.

Without schema type checking, the data grouping will output only one group with all schemas together since Q_1 only contains an overlapping attribute C, which is observed in all the groups.

6.3.2 Group Selection

While data grouping merge the data into groups, some groups may still have insufficient observations, which may affect the result of hypothesis testing. Therefore, we consider those groups as loner groups, not participating in the CLUSTERINGHAC step in Algorithm MD_{hac}. The criterion to judge loner groups is to set a *loner threshold* N: If the frequencies of all attributes in a group are lower than N, we consider it as a loner group and all the schemas in this group as loners. In statistics, N is conventionally set to 5, which is the recommended value for χ^2 hypothesis testing [2]. In our experiment, we find setting N to 3 is enough to contain sufficient observations.

Example 32: Consider the four groups in Example 31, the multinoimal expressions of these four groups (Q_2, Q_3, Q_4, Q_5), (Q_6, Q_7), (Q_1) and (Q_8) are: (A:4, B:4, C:1, D:2, E:1), (C:2, F:2, G:1), (C:1) and (C:1,

H:1) respectively. If we set the threshold N to 2, then groups (Q_1) and (Q_8) are considered as loner groups.

6.3.3 Loner Handling

After the step of CLUSTERINGHAC, we classify the loners into the accomplished clusters. As a classification problem, we classify a loner schema Q into the cluster with the largest probability to observe it. Formally, given a schema Q of length n, we will classify Q into the cluster C_i with the highest $Pr(Q|\mathcal{M}_i, n)$.

Some loners may have zero probabilities for all clusters. Equation 6.1 shows that when an attribute A_j does not exist in a cluster C_i, $Pr(A_j|\mathcal{M}_i) = 0$. For a schema Q with attributes not in any cluster, all the probabilities $Pr(Q|\mathcal{M}_i, n)$ will be 0 and thus we cannot decide which cluster to classify it into. To avoid this problem, in this step, we set $Pr(A_j|\mathcal{M}_i)$ to a very small value ε instead of 0 if A_j is not observed in C_i. In our implementation, we set $\varepsilon = 10^{-3}$.

Example 33: Continue with Example 32, assume after HAC clustering, the two clusters cannot be merged. We name group (Q_2, Q_3, Q_4, Q_5) as C_1 and (Q_6, Q_7) as C_2. From Section 6.2.1, we know multinomial distribution of C_1 is (A:0.33, B:0.33, C:0.08, D:0.17, E:0.08) and of C_2 is (C:0.4, F:0.4, G:0.2).

Now we need to classify loners Q_1 and Q_8 into these two clusters. For Q_1, by applying Equation 6.2, we have $Pr(Q_1|\mathcal{M}_1, 1)$ = 0.08 and $Pr(Q_1|\mathcal{M}_2, 1)$ = 0.4. Therefore, Q_1 is put into cluster C_2. Similarly, Q_8 is also put into cluster C_2. The final result of this clustering is (Q_2, Q_3, Q_4, Q_5) and (Q_1, Q_6, Q_7, Q_8).

6.3.4 Time Complexity

We evaluate the time complexity of MD_{hac}, for each individual step. Assume we have n schemas in G clusters with totally m attributes. Also, we assume the longest length of one schema is a constant C. DATAGROUPING can be executed in $O(C^2n^2) = O(n^2)$ time since we need to compare one schema with all the remaining schemas to check the containment relationship in Corollary 2. GROUPSELECTION can be executed in $O(mn)$ time in that for each group, we need to check the attribute frequencies. CLUSTERINGHAC takes $O(n^2m)$ time because every time we combine two clusters C_k and C_l, we only need to recompute the similarities between the remaining clusters and the new cluster $C_{<k,l>}$. The similarities between other clusters are not changed. So each iteration takes $O(nm)$ time and there are at most n iterations. Hence, we have the $O(n^2m)$ upper bound for this step. LONERHANDLING takes $O(nmG)$ time because for each loner schema, we need to check G clusters with the computation of probability over at most m attributes. The final step BUILDHIERARCHY is similar to CLUSTERINGHAC and takes $O(G^2m)$ time. Therefore, the time complexity altogether is bounded by $O(n^2m)$.

6.4 Experiments

To evaluate the MD_{hac} algorithm, we test it with 8 domains of structured sources on the deep Web. We compare our model-differentiation based approach with likelihood [51], entropy (COOLCAT) [24] and context linkage (ROCK) [35] based approaches using HAC algorithm and analyze the results. Also, we show the domain hierarchy built by MD_{hac} and evaluate the influence of the loner threshold N on the clustering performance.

MD_{hac}	Af	Am	Bk	Cr	Ht	Jb	Mv	Mr
C_1	0	101	0	0	2	4	0	0
C_2	0	0	62	0	0	1	9	2
C_3	0	0	0	24	0	0	0	0
C_4	0	0	0	0	35	0	0	1
C_5	0	0	0	0	0	50	1	0
C_6	53	0	0	0	1	0	0	0
C_7	0	0	0	0	0	0	8	67
C_8	0	1	7	0	0	0	62	7

(a) Conditional entropy of MD_{hac}: 0.32.

LK_{hac}	Af	Am	Bk	Cr	Ht	Jb	Mv	Mr
C_1	0	100	0	0	2	8	0	0
C_2	0	0	62	0	0	1	7	2
C_3	0	0	0	0	35	6	0	1
C_4	0	0	0	0	0	0	56	5
C_5	0	2	7	0	0	0	10	2
C_6	0	0	0	0	0	0	7	67
C_7	53	0	0	0	1	0	0	0
C_8	0	0	0	24	0	40	0	0

(b) Conditional entropy of LK_{hac}: 0.42.

EP_{hac}	Af	Am	Bk	Cr	Ht	Jb	Mv	Mr
C_1	0	100	0	0	2	4	0	0
C_2	0	0	62	0	0	0	5	2
C_3	0	0	0	24	0	0	0	0
C_4	0	0	0	0	35	6	0	1
C_5	0	0	0	0	0	0	57	5
C_6	0	0	0	0	0	0	8	67
C_7	53	0	0	0	1	0	0	0
C_8	0	2	7	0	0	45	10	2

(c) Conditional entropy of EP_{hac}: 0.38.

CL_{hac}	Af	Am	Bk	Cr	Ht	Jb	Mv	Mr
C_1	34	0	0	0	1	0	0	0
C_2	19	0	0	0	0	0	0	0
C_3	0	99	7	0	2	7	1	1
C_4	0	1	62	24	0	1	4	1
C_5	0	0	0	0	35	21	0	1
C_6	0	0	0	0	0	26	1	0
C_7	0	0	0	0	0	0	70	42
C_8	0	2	0	0	0	0	4	32

(d) Conditional entropy of CL_{hac}: 0.61.

Figure 6.8: Comparison of four similarity measures in HAC.

6.4.1 Experiment Setup

We use the TEL-8 dataset in the UIUC Web Integration Repository [17] to test our clustering algorithm. For each source, we manually extract attributes from its query interface by extracting noun phrases, and then judge its corresponding domain. This is our ground truth of "correct" clustering. The reason we do not apply our work in [72] for interface extraction is that we want to isolate the clustering process to study and thus fairly evaluate the performance.

To measure the result of clustering, we adopt the *conditional entropy* introduced in [8]. For a given number of clusters G, the value of the conditional entropy is within the range from 0 to $\log G$, where 0 denotes the 100% correct clustering, $\log G$ denotes purely random clustering result, *i.e.*, the sources from every single domain are evenly distributed into all clusters. Thus, the closer the conditional entropy value is to 0, the better the result is.

6.4.2 Experimental Results

We design three suites of experiments. *First*, we compare our approach MD_{hac} with the three existing approaches: likelihood based approach (LK_{hac}), entropy based approach (EP_{hac}) and context linkage based approach (CL_{hac}) for clustering the sources of 8 domains. For fair comparison, we only replace the similarity measure of test of model difference (Equation 6.6) in the CLUSTERINGHAC step with the likelihood based measure, entropy based measure and context linkage based measure. All the rest settings (pre-clustering and post-classification etc.) stay the same and the loner threshold N is set to 3.

To make the other measures clear, we briefly list each of them below. Reference [51] introduces the likelihood based similarity measure for HAC algorithm as Equation 6.8. The basic idea is that in each merging step in HAC, the two clusters generating the maximal likelihood after merging will be merged.

$$s(k,l) = \mathcal{L}(C_k) + \mathcal{L}(C_l) - \mathcal{L}(C_{<k,l>}). \tag{6.8}$$

COOLCAT [24] introduces entropy as the objective function, from where we derive the following similarity measure for HAC algorithm, with the same idea as the derivation of Equation 6.8 in [51].

$$s(k,l) = |C_k|E(C_k) + |C_l|E(C_l) - |C_{<k,l>}|E(C_{<k,l>}). \tag{6.9}$$

ROCK [35] introduces context linkage as the similarity measure:

$$s(k,l) = \frac{link[C_k, C_l]}{(n_k + n_l)^{(1+2f(\theta))} - n_k^{(1+2f(\theta))} - n_j^{(1+2f(\theta))}}. \tag{6.10}$$

The result in Figure 6.8 shows the comparison of the four measures in HAC algorithm. In particular, we present the results as the numbers of Web sources in each cluster from each domain. For example, in Figure 6.8 (a), 101 stands for that there are, in cluster C_1, 101 Web sources from automobile domain. We use the abbreviations Af, Am, Bk, Cr, Ht, Jb, Mv and Mr to denote the 8 domains Airfares, Automobiles, CarRentals, Hotels, Jobs, Movies and MusicRecords respectively. Figure 6.8 illustrates two results: 1) It is feasible to address the clustering of structured sources as the clustering of query schemas. The matrix of MD_{hac}, LK_{hac} and EP_{hac} do show correct clustering for most data. The result of (CL_{hac}) is not good perhaps because its similarity measure may not fit the schema data well; 2) MD_{hac} achieves, on clustering Web schemas, the best performance (smallest conditional entropy) among all the measures. In particular, compared with the second best measure, EP_{hac}, MD_{hac} has better clustering results for Jobs and Movies.

Second, we show the effectiveness of MD_{hac} to build the domain hierarchy. After clustering 8 domains, we continue with the BUILDHIERARCHY step to build the domain hierarchy in the same way as the HAC clustering. The result in Figure 6.9 illustrates that Automobiles and Jobs are merged in the same subtree, MusicRecords, Books and Movies in another subtree, and Airfares, CarRentals and Hotels in a third subtree. This hierarchy is consistent with our observation in the real world (*i.e.*, object domains are characterized by their query schemas): Books, MusicRecords and Movies are all media and often sold together online, and so are Airfares, CarRentals and Hotel reservations. Automobiles and Jobs are together because they share many location information, such as city, state and zip code.

Finally, we design experiments to evaluate the influence of the loner threshold N. In statistics, 5 is the recommended for the accuracy of χ^2 hypothesis testing and therefore N does not need to be larger than 5. We let N range from 2 to 5 and test all the four measures (We exclude $N = 1$ because $N = 1$ means no group selection). The result in Figure 6.10 shows that the clustering result is not affected too

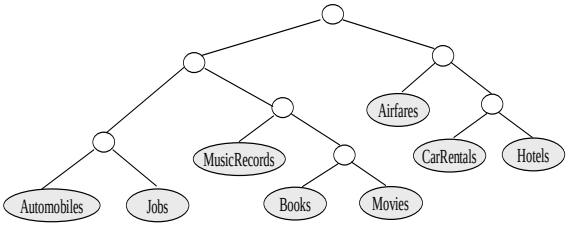


Figure 6.9: The domain hierarchy built by MD_{hac}.

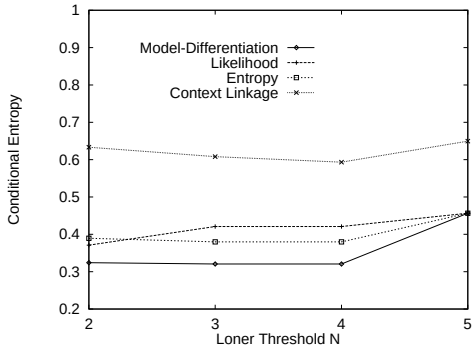


Figure 6.10: The influence of loner threshold N.

much by N, when N is ranged from 2 to 4. When $N = 5$, the result is worse because of the limited sampling size of our dataset. Setting N to 5 will trim most groups in group selection, where some insufficiently observed domains (*e.g.*, CarRental) are entirely trimmed out. Hence, we expect the result of $N = 5$ will be good when we have more observations. Putting in other words, when we have sufficient observations, the setting of N will not affect the result significantly.

6.5 Conclusion

This chapter studies the problem of organizing structured sources on the Web. Motivated by our observations of the deep Web, we propose to organize sources by their query schemas, and further abstract the problem as the clustering of categorical data. We develop a new model-differentiation objective function for clustering. Guided by the MD objective, we derive a new similarity measure for the general HAC algorithm. To apply statistical hypothesis testing for clustering, we design pre-clustering and

post-classification techniques. Our experiments show the effectiveness of our abstraction– By clustering the query schemas, we can accurately organize sources into object domains. Also, we show that the model-differentiation function outperforms existing ones with the hierarchical agglomerative clustering algorithm.

Chapter 7

Related Work

Schema matching (which this thesis mainly focuses on) is one critical step for schema integration [6, 61]. As a complete solution to automate the processing of matching Web forms, this thesis presents both the new idea of holistic schema matching (*i.e.*, the MGS and DCM frameworks and the ensemble scheme to maintain the matching quality with noisy input) and two steps to fully automate the matching process (*i.e.*, Web form crawling and clustering approaches). In this chapter, we thus accordingly organize related work with respect to each individual subproblem: schema matching, Web crawling and source clustering.

7.1 Schema Matching

We relate our holistic schema matching idea (from Chapter 2 to Chapter 4) to existing work in three aspects: the paradigms, the techniques and the input data.

Paradigms: Traditionally, schema matching relies on matchings between pairwise attributes before integrating multiple schemas. For instance, traditional binary or *n*-ary [55] schema integration methodologies (as [6] surveys) exploit pairwise-attribute correspondence assertions (mostly manually given)

for merging two or some n sources. Recent work on automatic schema matching mostly focuses on matchings between two schemas (e.g., [28, 50, 52, 47]). Therefore, the latest survey [60] abstracts schema matching as pairwise similarity mappings between two input sources. In contrast, we propose a new paradigm, holistic schema matching, to match many sources at the same time and discover all the matchings at once. Our work was motivated by integrating the deep Web, where the challenge of large scale matching is pressing. Our framework leverages such scale to enable statistical analysis.

Further, existing schema matching work mostly focuses on simple 1:1 matchings [28, 50, 52]. Complex matching has not been extensively studied, mainly due to the much more complex search space of exploring all possible combinations of attributes. Consider two schemas with u and v attributes respectively, while there are only $u \times v$ potential 1:1 matchings, the number of possible m:n matchings is exponential. The recent work iMAP [47] proposes to construct 1:n matchings between two schemas by combining their simple 1:1 matchings. Our DCM framework (Chapter 3) also aims at finding complex matchings. Although both aiming at complex matchings, our work is different from iMAP in: 1) scenario: iMAP focuses on matching two schemas, while we targets at large scale schema matching. 2) techniques: iMAP relies on the availability of instance values to construct complex matchings from simple 1:1 matchings, while we explore the co-occurrence information across schemas and thus develop a correlation mining approach.

The closest idea to the holistic matching paradigm is probably the corpus-based schema matching approach [36, 49], which suggests to use a separately-built schema corpus as a "knowledge-base" for assisting matching of unseen sources. While sharing the same insight of statistics analysis over corpora, our approach differs in that it leverages input schemas themselves as the corpus and assumes a generative model to unify the corpus.

Techniques: Many solutions have been developed to facilitate schema matching in automatic or semi-automatic way. The survey of [60] presents a taxonomy and comparison of these approaches. It classifies the solutions according to whether they deal with data values (called "instances") or schemas, whether the schema is flat or structured, and other aspects. There are many different techniques: Some approaches apply machine learning techniques to match a data source to the mediated schema such as the LSD system [28]. Some approaches use the structural similarity between schemas to find matchings, such as the flooding similarity matcher [52]. Cupid [50] presents a generic matching operation across different data models and applies a hybrid approach by combining both linguistic and structural similarity measurements.

In contrast, based on our observation of deep Web sources, we develop two statistical frameworks, MGS and DCM, which contrasts with existing techniques such as machine learning [28], constraint-based [46], structure-based [52], and hybrid approaches [50]. In the MGS framework, we hypothesize the existence of a hidden generative model for each domain (*e.g.*, Books, Movies). Under this hypothesis, a schema can be viewed as an instance generated from the model with some probabilistic behavior. Schema matching is thus transformed into the discovery of the hidden model, given a set of schema instances. In the DCM framework, we observe that co-occurrence patterns across schemas often reveal the complex relationships of attributes, which motivates us to abstract the problem of finding complex matchings as a dual mining of positive and negative correlations.

Further, to make the holistic matching framework robust against noise, we integrate it with an *ensemble* scheme, which aggregates a multitude of holistic matchers to achieve robustness, by exploiting statistical sampling and majority voting. We note that, our ensemble idea is inspired by *bagging classifiers* [11, 26] in machine learning. Bagging is a method for maintaining the robustness of "unstable" classification algorithms where small changes in the training set result in large changes in prediction.

In particular, it creates multiple versions of a classifier, trains each classifier on a random redistribution of the training set and finally takes a plurality voting among all the classifiers to predict the class. Therefore, our ensemble approach has the same foundation as bagging classifiers on exploiting majority voting to make an algorithm robust against outlier data in the input.

However, our approach is different from bagging classifiers in several aspects. First, *setting*: We apply the idea of the ensemble of randomized data for unsupervised learning (*e.g.*, in our scenario, schema matching with statistical analysis), instead of supervised learning (*i.e.*, human experts give the learner direct feedback about the correctness of the performance [45]), which bagging classifiers is developed for. Second, *techniques*: Our concrete techniques are different from bagging classifiers. In particular, in the sampling part, we take a downsampling other than random redistribution with replacement; in the voting part, we need to aggregate a set of ranked lists, which is more complicated than aggregate a set of labels in bagging classifiers. Third, *analytic modeling*: We build an analytic modeling specific to our matching scenario, which enables us to validate the effectiveness of a particular configuration and thus can be the basis for the design of the ensemble scheme.

Input Data: The previous work assumes their input as either relational or structured schemas. Those schemas are designed internally for developers. As a consequence, the attributes of the schemas may be named in a highly inconsistent manner, imposing many difficulties in schema matching. In contrast, our work focuses on matching query interfaces of deep Web sources. These interfaces are designed for end users and are likely more meaningful and consistent. Thus, we observed this distinguishing characteristic of "converging vocabulary" in our deep Web studies, which motivated our statistical approach.

Some recent works are particularly focusing on matching Web databases [41, 66, 64]. WISE [41] is a comprehensive query interface integrator, which evaluates the similarity of attributes in multiple aspects. However, it only deals with simple 1:1 matchings. Reference [64] matches query interfaces

based on the results of probing some instance values from the back-end databases via interfaces. It also only deals with simple 1:1 matchings. Comparing with other matching approaches, probing-based matching is much more expensive due to the large number of HTTP requests sent for each interface. In addition, it needs global model for each domain and is thus less scalable as an automatic generic solution for handling various domains of Web sources. Reference [66] pursues a clustering-based approach to discover 1:*n* matchings by exploring the "bridging" effect among query interfaces. However, its discovery of complex matchings essentially depends on a "hierarchical" interface extractor– That is, the grouping of attributes (*e.g.*, the grouping of last name and first name) must be identified, in the first place, by the interface extractor (and not the matching algorithm). This "hierarchy-recognition" requirement makes interface extraction a very challenging task.

In contrast, our matching algorithms only requires an interface extractor to extract a query interface as a "flat" set of query conditions, instead of a hierarchy of attributes, which can thus be easily satisfied (*e.g.*, our recent work of automatic interface extraction [72] is such an extractor). In fact, even with a simple "flat" extractor, it already introduces enough errors to impact the matching performance. In this thesis, we study such impact and propose an ensemble approach for maintaining the robustness of matching, which significantly extends the holistic matching idea (Chapter 4).

7.2 Web Crawling

We build a taxonomy to relate our crawling work in Chapter 5 to other Web crawling work. In particular, we present a taxonomy of 4-quadrant of Web crawlers in Figure 7.1, which contains two dimensions. Along the dimension of *subject topics*, crawlers are either topic-neutral on *any* topic or specific to *certain* topics. Along the dimension of *crawling target*, while traditional crawlers collect HTML pages, new type of crawlers collect certain Web artifacts (which we call objects), such as Web sites, query

		Crawling Target	
		Page	*Object*
Subject Topics	*Any*	General [13,21]	Object-Focused
	Certain	Topic-Focused [15,27,54,62]	Topic&Object-Focused [30,5]

Figure 7.1: The taxonomy of Web crawlers.

forms, products (*e.g.*, digital cameras), or addresses. This taxonomy thus classifies Web crawlers into four categories. For example, the traditional link-following crawlers [13, 21] fall into the category of topic-neutral and page-targeting crawlers, and the category of topic-focused crawlers [15, 27, 54, 62] look for pages on give topics.

The focus of our crawler work is crawling certain type of objects. A recent work [30] describes a crawler for collecting Websites related to specific topics. Given its target objects, Websites, it is natural for [30] to crawl site by site. Thus it can be viewed as a subset of the framework of our Site Finder. The work closest to ours is [5], in which a crawler is developed to find query forms on given topics. While the kind of objects targeted by [30] is Website, the target object in [5] and our work is Web form. Note that both [30] and [5] belong to the category of topic-focused and object-focused crawlers. By exploiting content-driven techniques (*e.g.*, content classifiers), they are not applicable in finding topic-neutral query forms.

In contrast, to the best of our knowledge, ours is the first attempt of building a topic-neutral object-focused Web crawler. Building upon the insight of structure-driven crawling, this new framework on the one hand eliminates the reliance on content focus, and on the other hand enables us to balance between high harvest (as virtually all the traditional crawlers focus on) and coverage, by exploiting the object distribution pattern in structure locality.

7.3 Source Clustering

We relate our clustering work in Chapter 6 to the literature in three aspects: in terms of the Web clustering problem and our clustering technique.

First, in terms of the *Web clustering problem*, existing Web clustering works mainly focus on clustering Web documents by exploiting Web content and linkage information [70, 71, 65, 42]. In contrast, our work focuses on the clustering of structured Web sources. With the observation that query schemas are discriminative representatives of sources, we are able to translate the original problem of source organization into the clustering of query schemas, a type of categorical data.

Second, in terms of the *clustering technique*, our work proposes a model-differentiation objective function for clustering query-schema data. Clustering of general categorical data has recently been more actively studied, *e.g.*, STIRR [34], CACTUS [33], ROCK [35], and COOLCAT [24]. STIRR treats clustering as a partitioning problem of hypergraph and solves it based on non-linear dynamical systems. CACTUS considers a cluster as a set of pairwise strong connected attributes by measuring attribute occurrences. ROCK, COOLCAT and this work pursue the same direction of defining a new similarity measure involving the *global context* (such as properties of a entire cluster) instead of local pairwise measure. ROCK uses context linkages between data points, and COOLCAT uses entropy of clusters. As an alternative, we develop the model-differentiation measure, which maximizes the statistical heterogeneity among clusters.

Our statistical approach belongs to the general idea of model-based clustering (*e.g.*, partitional EM algorithm [53] and hierarchical algorithms [4, 32]). Such clustering assumes that data is generated from a mixture of distributions, each of which defines a cluster. This general approach is traditionally not specific to categorical data– More recently, reference [51] proposes a multivariate multinomial distribution (in which each feature is an independent multinomial distribution) for categorical data. In comparison,

the model we propose for schema data (or transactional data) is a "joint" multinomial, where all features are generated from a multinomial distribution.

All the existing model-based works essentially use likelihood as the objective function to maximize– In contrast, we propose model-differentiation by maximizing the statistical heterogeneity among clusters. In our extended report [40], we show that these two objective functions are in fact *equivalent* in assessing the global clustering results. However, toward their "global" objectives, they indeed imply *different* greedy "local" similarity measures. In our experiments, we also compare the model-differentiation measure with the likelihood one on HAC algorithm.

Chapter 8

Conclusion

This thesis proposes to move the traditional pairwise attribute correspondence toward a new holistic paradigm in the discovering of semantic matchings among attributes. This holistic approach is well suited for the new frontier of massive networked databases, such as the deep Web. As the realizations of the holistic schema matching, we develop the MGS and DCM frameworks in sequence with global and local evaluation strategies respectively.

On the one hand, global evaluation is a systematic and principled way to evaluate models since it exhaustively evaluates all possible models with a statistical basis. In particular, in the MGS framework, statistical hypothesis testing can report matchings with respect to a given theoretical *significance level*. Also, the discovered model can naturally be employed as a unified schema to mediate queries to specific sources. However, global evaluation can be expensive. The exploration of all the possible models can be generally exponential. Further, modeling can be a difficult task, depending on specific target semantics to be discovered. In particular, it is unclear how to extend the modeling in Chapter 2 to accommodate complex matchings, which the DCM framework copes with.

On the other hand, local evaluation adopts a greedy strategy to incrementally construct a potentially suboptimal model. The greedy selection is not as systematic as the exhaustive enumeration in the global evaluation. Also, as the core of correlation mining, we need to choose an appropriate correlation measure for our application scenario. Since the correlation measure is often empirically designed based on heuristics, the mining result may lack a principled justification. However, our experiments show that the matching accuracy of local evaluation is empirically good enough in discovering both simple and complex matchings. Further, local evaluation has some other advantages that global evaluation does not have: First, the computation of local evaluation is very efficient, since instead of exhaustively exploring every model as a whole, we select one matching at a time as part of the best model. Second, it is easier to accommodate complex matchings in local evaluation since it does not require formal statistical modeling. In particular, the DCM framework supports complex matchings by considering both positive and negative correlations. Given the respective strengthes and weaknesses of global and local evaluations, we wonder if a hybrid of the two approaches will achieve the strength of both without the weakness of either.

Further, to complete the automatic process of holistic schema matching, we also address two other related issues: How to maintain the robustness of a holistic matcher when the input schemas contain errors and how to organize schemas into their corresponding domains.

First, to make holistic matching approaches robust to noisy input from the automatic interface extractor, we integrate a holistic matcher with an *ensemble* scheme, which aggregates a multitude of the matchers to achieve robustness, by exploiting statistical sampling and majority voting. In this thesis, we apply such an ensemble scheme for the DCM matcher and our empirical study shows that the ensemble approach can significantly boost the matching accuracy under noisy schema input, and thus maintain the desired robustness of the DCM matcher.

Second, to obtain a set of schemas in the same domain (as the input of our holistic matching algorithms), we develop techniques for source discovery (*i.e.*, automatically finding large scale query interfaces on the Web) and schema clustering (*i.e.*, automatically organize discovered schemas into a domain hierarchy).

In particular, in source discovery, we aim at building a crawler for collecting query forms on the Web. We abstract this problem as object-focused, topic-neutral crawling and propose a structure-driven crawling framework for such a crawling task by observing the existence of structure locality of query forms. We develop the Web Form Crawler to realize the framework. The experimental results show that our crawler can not only maintain stable harvest but also steadily grow coverage throughout the crawling. Compared to page-based crawling, our best harvest rate is about 10 to 400 times difference, depending on the page traversal schemes used.

In schema clustering, we propose a new model-differentiation objective function for clustering. Guided by the MD objective, we derive a new similarity measure for the general HAC algorithm. To apply statistical hypothesis testing for clustering, we design pre-clustering and post-classification techniques. Our experiments show the effectiveness of our abstraction. Also, we show that the model-differentiation function outperforms existing ones with the hierarchical agglomerative clustering algorithm.

Bibliography

[1] R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. In *SIGMOD Conference*, 1993.

[2] A. Agresti. *Categorical Data Analysis*. John Wiley & Sons, Inc. New Jersey, 2002.

[3] D. R. Anderson, D. J. Sweeney, and T. A. Williams. *Statistics for Business and Economics (Second Edition)*. West Pub. Co., 1984.

[4] J. D. Banfield and A. E. Raftery. Model-based gaussian and non-gaussian clustering. *Biometrics*, 49(3):803–821, 1993.

[5] L. Barbosa and J. Freire. Searching for hidden-web databases. In *WebDB Workshop*, pages 1–6, 2005.

[6] C. Batini, M. Lenzerini, and S. B. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18(4):323–364, 1986.

[7] M. K. Bergman. The deep web: Surfacing hidden value. Technical report, BrightPlanet LLC, Dec. 2000.

[8] P. Berkhin. Survey of clustering data mining techniques. Technical report, Accrue Software, San Jose, CA, 2002.

[9] P. J. Bickel and K. A. Doksum. *Mathematical Statistics: Basic Ideas and Selected Topics*. Prentice Hall, 2001.

[10] J. C. Borda. Mémoire sur les élections au scrutin. *Histoire de l'Académie Royale des Sciences*, 1781.

[11] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[12] S. Brin, R. Motwani, and C. Silverstein. Beyond market baskets: generalizing association rules to correlations. In *SIGMOD Conference*, 1997.

[13] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

[14] H. D. Brunk. *An Introduction to Mathematical Statistics*. New York, Blaisdell Pub. Co., 1965.

[15] S. Chakrabarti, M. van der Berg, and B. Dom. Focused crawling: a new approach to topic-specific web resource discovery. In *WWW Conference*, 1999.

[16] K. C.-C. Chang, B. He, C. Li, M. Patel, and Z. Zhang. Structured databases on the web: Observations and implications. *SIGMOD Record*, 33(3), 2004.

[17] K. C.-C. Chang, B. He, C. Li, and Z. Zhang. The UIUC web integration repository. Computer Science Department, University of Illinois at Urbana-Champaign. http://metaquerier.cs.uiuc.edu/repository, 2003.

[18] K. C.-C. Chang, B. He, and Z. Zhang. Toward large scale integration: Building a metaquerier over databases on the web. In *CIDR Conference*, 2005.

[19] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani. Robust and efficient fuzzy match for online data cleaning. In *SIGMOD Conference*, 2003.

[20] J. Cho and H. Garcia-Molina. Parallel crawlers. In *WWW Conference*, 2002.

[21] J. Cho, H. García-Molina, and L. Page. Efficient crawling through URL ordering. *Computer Networks and ISDN Systems*, 30(1–7):161–172, 1998.

[22] J. Cope, N. Craswell, and D. Hawking. Automated discovery of search interfaces on the web. In *Proc. of the 14th Australasian Database Conference*, pages 181–189, 2003.

[23] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. Chapter 34. In *Introduction to Algorithms (Section Edition)*, 2001.

[24] B. D., C. J., and L. Y. Coolcat: An entropy-based algorithm for categorical clustering. In *11th International Conference on Information and Knowledge Management*, pages 582–589, 2002.

[25] P. Diaconis and R. Graham. Spearman's footrule as a measure of disarray. *Journal of the Royal Statistical Society, Series B*, 39(2):262–268, 1977.

[26] T. G. Dietterich. Machine-learning research: Four current directions. *The AI Magazine*, 18(4):97–136, 1998.

[27] M. Diligenti, F. Coetzee, S. Lawrence, C. L. Giles, and M. Gori. Focused crawling using context graphs. In *VLDB Conference*, 2000.

[28] A. Doan, P. Domingos, and A. Y. Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In *SIGMOD Conference*, 2001.

[29] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation methods for the web. In *WWW 2001 Conference*, 2001.

[30] M. Ester, H.-P. Kriegel, and M. Schubert. Accurate and efficient crawling for relevant websites. In *VLDB*, pages 396–407, 2004.

[31] R. Fagin, R. Kumar, and D. Sivakumar. Efficient similarity search and classification via rank aggregation. In *SIGMOD 2003 Conference*, 2003.

[32] C. Fraley. Algorithms for model-based Gaussian hierarchical clustering. *SIAM Journal on Scientific Computing*, 20(1):270–281, 1999.

[33] V. Ganti, J. Gehrke, and R. Ramakrishnan. CACTUS - clustering categorical data using summaries. In *Knowledge Discovery and Data Mining*, pages 73–83, 1999.

[34] D. Gibson, J. M. Kleinberg, and P. Raghavan. Clustering categorical data: An approach based on dynamical systems. *VLDB Journal*, 8(3–4):222–236, 1998.

[35] S. Guha, R. Rastogi, and K. Shim. ROCK: A robust clustering algorithm for categorical attributes. *Information Systems*, 25(5):345–366, 2000.

[36] A. Halevy, O. Etzioni, A. Doan, Z. Ives, J. Madhavan, L. McDowell, and I. Tatarinov. Crossing the structure chasm. *Conf. on Innovative Database Research*, 2003.

[37] B. He and K. C.-C. Chang. Statistical schema matching across web query interfaces. In *SIGMOD Conference*, 2003.

[38] B. He, K. C.-C. Chang, and J. Han. Automatic complex schema matching across web query interfaces: A correlation mining approach. Technical Report UIUCDCS-R-2003-2388, Dept. of Computer Science, UIUC, Dec. 2003.

[39] B. He, K. C.-C. Chang, and J. Han. Discovering complex matchings across web query interfaces: A correlation mining approach. In *SIGKDD Conference*, 2004.

[40] B. He, T. Tao, and K. C.-C. Chang. Clustering structured web sources: A schema-based, model-differentiation approach. In *EDBT'04 ClustWeb Workshop*, 2004.

[41] H. He, W. Meng, C. Yu, and Z. Wu. Wise-integrator: An automatic integrator of web search interfaces for e-commerce. In *VLDB 2003 Conference*, 2003.

[42] X. He, H. Zha, C. Ding, and H. Simon. Web document clustering using hyperlink structures. Technical Report CSE-01-006, Dept. of Computer Science and Engineering, Pennsylvania State University, 2001.

[43] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3):264–323, 1999.

[44] J. G. Kemeny. Mathematics without numbers. *Daedalus*, 88:571–591, 1959.

[45] P. Langley. *Elements of Machine Learning*. Morgan Kaufmann, 1995.

[46] J. Larson, S. Navathe, and R. Elmasri. A theory of attributed equivalence in databases with application to schema integration. *IEEE Trans. on Software Engr.*, 16(4):449–463, 1989.

[47] Y. Lee, A. Doan, R. Dhamankar, A. Halevy, and P. Domingos. imap: Discovering complex mappings between database schemas. In *SIGMOD Conference*, 2004.

[48] Y.-K. Lee, W.-Y. Kim, Y. D. Cai, and J. Han. Comine: Efficient mining of correlated patterns. In *Proc. 2003 Int. Conf. Data Mining*, Nov. 2003.

[49] J. Madhavan, P. Bernstein, A. Doan, and A. Halevy. Corpus-based schema matching. In *ICDE Conference*, 2005.

[50] J. Madhavan, P. A. Bernstein, and E. Rahm. Generic schema matching with cupid. In *VLDB Conference*, 2001.

[51] M. Meilă and D. Heckerman. An experimental comparison of several clustering and initialization methods. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, pages 386–395, 1998.

[52] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *ICDE 2002 Conference*, 2002.

[53] T. M. Mitchell. *Machine Learning*. MIT Press,McGraw-Hill, 1997.

[54] S. Mukherjea. Wtms: A system for collecting and analyzing topic-specific web information. In *WWW Conference*, 2000.

[55] S. Navathe and S. Gadgil. A methodology for view integration in logical data base design. In *VLDB*, 1982.

[56] E. Omiecinski. Alternative interest measures for mining associations. *IEEE Trans. Knowledge and Data Engineering*, 15:57–69, 2003.

[57] OpenDirectoryProject. DMOZ site list. http://rdf.dmoz.org/rdf/content.rdf.u8.gz.

[58] J. Ponte and W. Croft. A language modelling approach to information retrieval. In *Proceedings of the 21st ACM SIGIR Conference on Research and Development in Information Retrieval*, 1998.

[59] M. Porter. The porter stemming algorithm. Accessible at http://www.tartarus.org/˜martin/PorterStemmer.

[60] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4):334–350, 2001.

[61] L. Seligman, A. Rosenthal, P. Lehner, and A. Smith. Data integration: Where does the time go? *Bulletin of the Tech. Committee on Data Engr.*, 25(3), 2002.

[62] S. Sizov, M. Theobald, S. Siersdorfer, G. Weikum, J. Graupmann, M. Biwer, and P. Zimmer. The BINGO! system for information portal generation and expert web search. In *CIDR*, 2003.

[63] P. Tan, V. Kumar, and J. Srivastava. Selecting the right interestingness measure for association patterns. In *ACM SIGKDD Conference*, July 2002.

[64] J. Wang, J.-R. Wen, F. Lochovsky, and W.-Y. Ma. Instance-based schema matching for web databases by domain-specific query probing. In *VLDB 2004 Conference*, 2004.

[65] Y. Wang and M. Kitsuregawa. Evaluating contents-link coupled web page clustering for web search results. In *Proceedings of the 7th International Conference on Information and Knowledge Management*, 2002.

[66] W. Wu, C. T. Yu, A. Doan, and W. Meng. An interactive clustering-based approach to integrating source query interfaces on the deep web. In *SIGMOD Conference*, 2004.

[67] YahooSearchBlog. http://www.ysearchblog.com/archives/000172.html.

[68] H. P. Young. An axiomatization of borda's rule. *J. Economic Theory*, 9:43–52, 1974.

[69] H. P. Young. Condorcet's theory of voting. *American Political Science Review*, 82:1231–1244, 1988.

[70] O. Zamir and O. Etzioni. Web document clustering: A feasibility demonstration. In *SIGIR Conference*, pages 46–54, 1998.

[71] O. Zamir, O. Etzioni, O. Madani, and R. M. Karp. Fast and intuitive clustering of web documents. In *Knowledge Discovery and Data Mining*, pages 287–290, 1997.

[72] Z. Zhang, B. He, and K. C.-C. Chang. Understanding web query interfaces: Best-effort parsing with hidden syntax. In *SIGMOD Conference*, 2004.