PROBABILISTIC VISUAL RELATIONAL DATA EXTRACTION

BY

NGOC TRUNG BUI

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2010

Urbana, Illinois

Adviser:

Associate Professor Kevin Chen-Chuan Chang

Abstract

This paper studies the problem of wrapper generation and proposes the concept of visual-relational data extraction as the foundation for modeling wrappers. Towards large scale integration, we identify the key requirements of wrapper deployment, and observe the limitations of the state of the art— which inherently result from their low-level wrapper modeling. We thus propose the visual-relational modeling and develop the execution and learning mechanisms. Our experiments show significant improvements towards satisfying the accuracy and consistency requirements.

To my Father and Mother.

Acknowledgments

This project would not have been possible without the support of many people. Many thanks to my advisor, prof. Kevin Chen-Chuan Chang, who read my numerous revisions and helped me to make the approach clear. Thanks to the Vietnam Education Foundation for providing me with the financial means to complete this project. And finally, thanks to my parents, and numerous friends who endured this long process with me, always offering support and love.

Table of Contents

List of Figures

List of Abbreviations

DI	Data Integration.
DE	Data Extraction.

List of Symbols

Ω	Visual Model.
Υ	Data Record.
e	Data Token.

Chapter 1

Introduction

Wrapper generation is fundamental for enabling data extraction from structured data sources, a crucial step in information integration and search. This paper attempts to consider wrapper generation with a new paradigm of modeling data sources. While various approaches exist, they all uniformly resort to HTML features and tag patterns to model the "regularity" of sources. We observe that modeling is fundamental—which inherently limit existing approaches to match several key requirements. As a different approach, we propose *visual relational* modeling, which aim to specify wrappers with high-level features and only minimal patterns.

While a well-recognized problem, with the prevalence of databases on the Web, wrapper generation is increasingly a barrier for realizing large scale information integration across the Internet. On this "deep Web," numerous data sources provide structured information (e.g., amazon.com for books; cars.com for automobiles) accessible only via dynamic queries instead of static URL links. To explore the contents behind the surface from such databases, as a major hurdle, we must extract structured data from the query results—which we refer to as *data pages*. To illustrate, Figure 1.1 shows two data pages from Yahoo, at different times. Such data pages presents a set of *records*, *e.g.*, [jobtitle, company, location, date], which are dynamically retrieved from the underlying database.

With the proliferation of databases on the Web, users' need to access such information has been pressing and, consequently, wrapper generation has become the key enabling techniques. Current search services cannot meaningfully index such data, precisely due to the challenge of extracting data from HTML text pages. With effective wrapper construction, we will be able to enable large scale integration of specialized and structured information, *e.g.*, building *vertical search* over various structured domains such as *jobs* (*e.g.*, simplyhired.com crawls and extracts job data from thousands of company sources) and *shopping* (*e.g.*, thefind.com indexes product information from numerous vendors).

In practical deployment towards building large scale vertical search, however, we realized that current wrapper approaches fall short in several critical aspects. To motivate, we systematically examine the full life cycle of a wrapper, towards scalable and cost-effective wrapper deployment (Chapter 3). While we

Featured Job Results (5,360) What's this?

View: **Brief** | Detailed — Sorted by: **Top Results** — **1-30** out of **6,171 Jobs**

Job Title	Company Name	Location	Date
Senior Consultant - Business Intelligence - Save Job	CyberCoders	Denver, CO	New Today
Business Intelligence Specialist - SQL, Java, Crystal Report - Save Job	CyberCoders	Cleveland, OH	New Today
Senior Consultant - Business Intelligence - Save Job	CyberCoders	Minneapolis, MN	New Today
Business Intelligence Specialist - SQL, Java, Crystal Report	CyberCoders	Cleveland, OH	Aug 11

Sort by: **Most Relevant** | Date — 1-50 of over 1000 | First | Previous | Next | Last

Job Title	Company Name	Location	Salary	Date
Senior Consultant - Business Intelligence	BearingPoint	Chicago, IL	--	Oct 27
Senior Signal Intelligence (SIGINT)Engineer	Adecco Technical	Scottsdale, AZ	--	Oct 27
Financial Services and Business Intelligence	BearingPoint	Mountain View, CA	--	Oct 27
Intelligence and Communications	United States Navy	Atlanta, GA; Miami, FL; Denver, CO;	--	Oct 27

Figure 1.1: hotjobs.yahoo.com: two different dates.

identify three key requirements — *accuracy*,*consistency*, and *intuitiveness* — unfortunately, no existing approaches satisfy all. While their induction approaches differ (Chapter 2), they are essentially identical in their wrapper modeling, which rely on low-level HTML features and tag-sequence patterns, resulting in wrappers that require rigid regularity, fragile to changes, and unintuitive to understand.

As our key insight, we propose to *elevate* representation to visual perception and to *minimize* the patterns of wrappers to only relations between desired elements. Our proposal is guided by the "dual" principles of wrapper modeling: high-level features and minimal patterns. With visual-relational modeling as the core, we develop model execution for data extraction, and model induction for wrapper generation, thus completing the overall framework.

We have performed extensive experimental evaluation, and the results demonstrate significant improvement over existing approaches. For concrete and realistic study, we collected a large dataset, the *2Y5D Dataset*, over two years (October 2004 - August 2006) across five domains (Auto, Book, Job, Movie, Music). We compare our visual relational framework to several representative existing approaches. For accuracy, our system returns high F1-measure in the range of 85%-95%, outperfoming the second-best apporache by a margin of 20%-55%. For consistency over time, our system preserves wrapper correctness for far longer periods than existing approaches, in the range of 200%-700% times. We have deployed the system in building large scale vertical search for the apartment domain, which requires building agents for thousands of rental data sources, and our experience in the industrial setting has been encouraging and consistent with the experimental evaluation.

In summary, our contribution in this paper includes:

- *Concept*: we propose novel concept of **visual-relational data extraction** for wrapper modeling.
- *Framework*: We propose effective *execution* and *learning* of the visual-relational model.
- *Evaluation*: We extensively evaluated the accuracy and consistency of our approach over two years of real dataset.

Chapter 2

Related Work

Related works to ours are in wrapper induction research. We, therefore, want to compare with them in the following two aspects of this topic.

Wrapper Model: In term of *model language*, most of previous works [1, 2, 3, 4, 5, 6, 7] use low-level rules directly in HTML source code to skip unnecessary information and reach to specific pattern of desired information. Baumgartner et. al. [8] uses prolog-like language called *Elog*, which contains visual-based predicates such as *before, after*. However, these predicates actually reflex the internal order of HTML tag structure rather than on the interface level. The main problem with these low-level languages is the inconsistency of wrapper description in the context of rapidly updating and changing speed of webpages. To the best of our knowledge, our work is the first research work which completely leverage the wrapper model into visual abstraction level by using probabilistic visual relations. In term of *output structure* support, linear approaches [6, 9] support a linear extraction without optional, repetitive and nested structure. Hsu et. al. introduce non-linear finite-state transducer on SoftMealy to deal with missing and multi-value attribute. Hierarchical-model-based approaches in STALKER [3], RoadRunner [1] and XWRAP [10] support all kinds of attribute variation such as missing values, multi-value and nested structure. The relational visual model presented in this paper provides all of these supports.

Induction technique The very first approach focuses on developing some declarative languages to assist users in constructing wrappers. These languages are proposed as simpler alternatives for common functions written in general programming languages. Some systems belong to this approach are [11, 2]. Building rules in these supported languages is not intuitive and extremely error-prone to users.

Supervised learning approaches to learn data extraction rules and/or patterns. Later on, these rules and patterns are used to identify data elements that follow them and assign label. The usual accuracy of this approach is not very good since most data does not always follow the same rules and patterns. Many induction systems have been introduced include in [5, 3, 6]. In our approach, we only need one training example which require no expertise from users. Even with minimal input from users, our technique achieves a very high accuracy because most of the inconsistency has been removed when we leverage our system into

the highest level of abstraction. It is also worthy to know that in the context of this paper, we consider our approach as semi-automatic even though the implemented system is fully automatic with additional domain knowledge - in comparison with RoadRunner [1] needs at least two similar pages and ViNTs [7] requires both multi-record pages and non-result pages from search engines.

Automatic learning [1, 12, 13] base on the regularity of HTML structure as the basis for alignment and extraction. These methods, however, are not very robust since they require very structured input pages to have a good accuracy. Not to mention many of the pages different data record might have different tag structure because of their format different. Generally, the output of these approaches need to be intensively post-processed to be used. In our approach, we require minimal label training records (i.e. user just hight light what they want on one data record) to avoid post-processing and labeling. As noted above, the technique in this paper are comparable with automatic learning technique.

Visual Usage: In a different perspective, related works to our paper also contain papers which use visual information in extracting/alanyzing webpages [14, 7, 15, 16, 17]. Deng et. al. use visual alignment to identify the meaning of webpage regions such as banner, main content, menu, etc. Webform analyzing research [16, 17] also partially/fully use visual information in identifying form elements as well as associating with corresponding labels. ViPER [15] utilizes visual bounding box as the main measure in ranking data regions which helps to eliminate low-informative data regions in output. However, the extraction algorithm in ViPER (i.e. Global Sequence Alignment) is applied completely in HTML source code. ViNTs [7] has an interesting idea in introducing the visual block regularity in extraction. However, this method is not applicable in extracting detail attributes of each data record where the attributes are written in a sequence (e.g. book's attributes in Amazon.com) since the shapes of each data record is completely different. Moreover, the paper made a very strong assumption to have both resulted and non-resulted pages from search engine which virtually give the correct extracted regions. The technique in ViNTs is also trickly since it depends too much on multiple heuristics to identify first content line of a records.

Chapter 3

Motivation: Model Matters

Observation: Wrapper through Life Cycle. Let's start with observing the "big picture" for a wrapper in its full course of operation. In Figure 3.1, centering around a wrapper (the shaded box), there are several key stages of creation, execution, and maintenance.

• *Wrapper Creation & Repair:*. At the very first *wrapper creation* stage, a "wrapper developer" creates a wrapper for a source (*e.g.*, amazon.com books). Essentially, such creation will build a *wrapper model*, which we denote Ω, for specifying the template structure of the source for data extraction.

This stage has been the focus of most wrapper research–How to automate wrapper generation as much as possible? Many "mostly automatic" approaches have been developed, as Chapter 2 discussed. In particular, as a representative category, wrapper *induction* takes a few example pages from the source and automatically "induce" the underlying template as HTML tag tree patterns, which is then used as the model Ω for data extraction by recognizing the same tag patterns in future pages.

No current solutions are fully automatic; they all require certain amount of manual efforts—typically for *collecting* one or multiple training pages, *labeling* these pages, or *matching* the induced template slots to our desired data attributes. As an example, the RoadRunner system [] takes multiple pages in training, does not need labeling, but requires developers to check the output templates and select some slots as desired attributes (say, in the pattern `<li><i>Title:</i>` ... `<href> #pcdata </href>` ... `</li>`, the `#pcdata`" slot is for attribute title).

The stage also handles wrapper repairing. When a wrapper breaks, such as due to source changes, the developer will fix the wrapper, either by regenerating it from scratch (requiring collecting new training example pages, labeling, *etc.*) or by inspecting and fixing the model directly.

• *Wrapper Execution:*. In regular production, at the *wrapper execution* stage, we will use the wrapper to extract data records from input pages from the source. Essentially, the wrapper will execute its model Ω over each input page, *i.e.*, to match Ω (say, as tag tree patterns in RoadRunner) with the page and thus extract data in desired slots. Thus routinely, given data pages as input, the wrapper outputs extracted data,

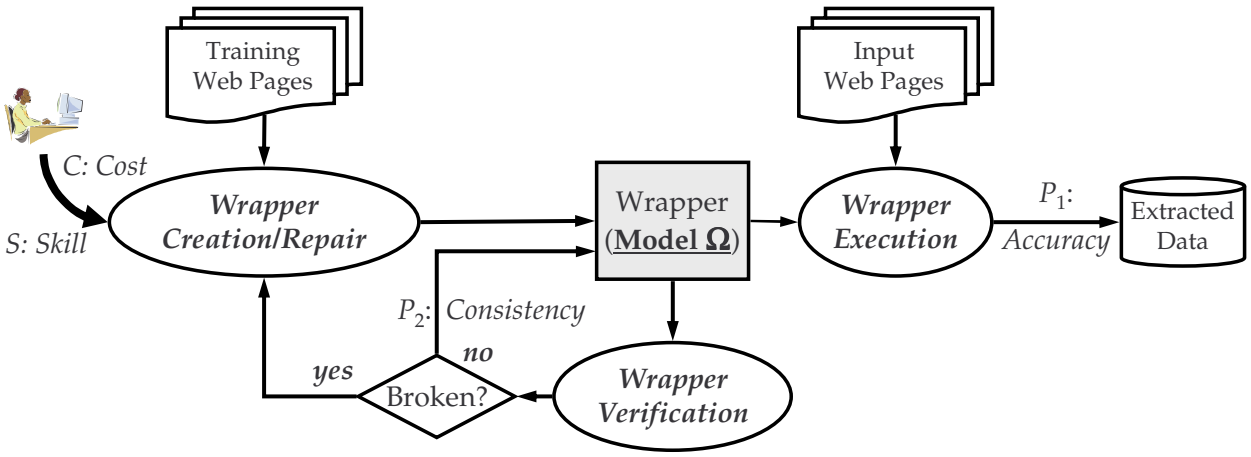

Figure 3.1: Wrapper through its full life cycle.

by executing its model trained earlier.

The exact execution (or "parsing") mechanism depends on how the model is expressed. For instance, in most induction approaches, when Ω uses tag tree path patterns, the wrapper find the matching paths (and data elements) from the DOM tree of an input page. If Ω uses tag delimiters, then the wrapper would locate the matching tags and identify data values in between.

• *Wrapper Verification:*. Over time, a wrapper may *break*—*i.e.*, it can no longer extract data satisfactorily from the source—since the source may change. When the source changes its page structure, the wrapper's model Ω does not match the source pages well any more. As such changes are expected, in the wrapper verification stage, we must regularly check the "health" of the wrapper, *e.g.*, by monitoring the quality of the output data. If the wrapper indeed breaks, it will be sent back to the first stage for repairing.

Not all source changes will break a wrapper. The exact impact depend on the particular model of the wrapper. Since different wrapper approaches use different model and execution mechanisms, they will differ in how their wrappers can react to changes. For instance, as most induction approaches resort to HTML tag path patterns, for any small change (say, by inserting an addition tag `<b>...</b>`, a path pattern may become mismatching.

Implications: Wrapper Requirements. Throughout the life cycle of a wrapper, we can clearly identify several important requirements for its effective operation. As the basis, Figure 3.1 marks the performance "parameters."

• Labor L: In creation-&-repair, how much manual labor work does it require?

• Cost S: In creation-&-repair, what skill does it require?

- Accuracy P_1: In execution, how accurate is the wrapper?
- Consistency P_2: In verification, how consist does the wrapper remain correct over time?

With these key parameters that characterize various aspects of a wrapper approach, we clearly identify the following requirements for a wrapper framework to be effective.

• **R1**: *Accuracy:*. To produce high quality data, we require high accuracy; *i.e.*, to maximize P_1. To achieve accuracy, a good framework must be robust in handling various sources with varying degrees of template "regularity" to induce.

• **R2**: *Consistency:*. To reduce maintenance cost, we require high consistency; *i.e.*, to maximize P_2. To achieve consistency, a good framework must be resistant to source evolutions with varying degrees of change significance. We stress that, with the rapid evolution of Web data, sources tend to change more and more frequently, and thus consistency is crucial.

• **R3**: *Intuitiveness:*. To reduce human cost, we require high intuitiveness of working with the framework; *i.e.*, to reduce sophisticated work, or L and S. Where is the manual work? To begin with, as just explained, full automation is unlikely, and most approaches require certain manual work in preparing the input and matching the output of wrapper creation. Further, as no such "automatic" approaches can guarantee 100% accuracy, a developer often needs to *correct* or *tune* a wrapper (including repairing broken wrappers). Thus, in addition to reducing the amount of work L, we also desire that the generated wrappers—or their models—are easy to understand by users.

Problems: Current Deficiencies. As we outline the requirements, we found that, unfortunately, no current approaches meet all the requirements. We discuss each requirement in turn. To be concrete, we use two example pages from hotjobs.yahoo.com, as Figure 1.1 shows, collected at two different dates (August 2005 and October 2004, respectively)—excerpted from our 2Y5D-Dataset (a set of pages over two years in 5 domains; Table 5.1.

First, for accuracy: Most current approaches require rigid regularity in HTML tag path sequences with a fundamental assumption that all data records share similar tag paths. Such assumption can often be violated with today's increasingly complex page styles and HTML coding, and thus compromise accuracy.

Consider a simple example in Figure 1.1b, where the odd and even rows (in the tabular listing) are of different formats, which are results of different underlying HTML tag values and tag structures. Thus the DOM subtrees of even and odd tuples can be quite different. This type of page, therefore, causes difficulties for current approaches that use HTML tag patterns—essentially because that the regularity at the HTML

level is limited. (Our experiments in Chapter 5 validate this observation by comparing the robustness of different approaches for different structures.)

Second, for consistency: All current approaches rely on quite "low-level" and "internal" page features in their modeling, which are rather sensitive to even small changes in sources. The existing framework all resort to HTML-level characteristics, such as DOM structure, color, text pattern, length of data, text size, *etc.*, as their features for modeling (the Ω). Those features are only seen in the HTML coding—and not visible to end users; thus they represent low-level and internal detail that may change, even when the desired elements are largely unaffected. Consequently, the current approaches compromise consistency, with their choice of model features.

For example, observe the two pages in Figure 1.1, which captures the evolution of the hotjobs.yahoo.com. While the visual characteristics are quite similar (*e.g.*, the attributes are aligned in the same way visually), their underlying HTML features are radically different, and will break any wrappers that remember such patterns. (Chapter 5 also validates this observation by comparing the consistency of different approaches over a two-year course.)

Third, for intuitiveness: With the low-level HTML features and tag path structure as their model expression "language," current wrappers require users who can speak HTML code. While everyone can browse Web pages, it requires relatively skilled programmers to manipulate HTML code. Thus, current approaches, again, compromise intuitiveness.

For instance, for patterns generated by say RoadRunner, the developer needs to match the data slots to attributes, which will require reading HTML code (and regular expressions) of `<li><i> Author: </i> (<br> #pcdata </b>)+`.

Insight: Model Matters. As we just analyzed, it becomes evident that the deficiencies of current state of the art are *inherently* due to the choice of *modeling*—*i.e.*, how we describe extraction patterns. While many approaches have been with different techniques, surprisingly, to date, they all uniformly assume HTML-level features and patterns as the modeling language. The low-level modeling has resulted in relying on rigid patterns (thus reducing accuracy), sensitive to internal and small changes (thus affecting consistency), and requiring HTML skill (thus barring intuitiveness).

Our main thesis in this paper is, therefore, the choice of modeling matters. We aim to address the current deficiencies by understanding the impact of modeling, and to propose an effective framework with novel modeling.

The Wrapper Modeling Principles. Reflecting on the limitation of current approaches, we believe that appropriate modeling must follow two principles:

- **High-level Features:** As just explained, current modeling relies on low-level HTML features that are internal to a page (or invisible to users), which are thus likely irregular and unstable. Our modeling should use "high-level" features that are visible to human users.
- **Minimal Patterns:** Further, current modeling also relies on regularity patterns that involve tag sequence that are either *paths* leading to the desired elements or *delimiters* around them. Such patterns tend to be compromised by even changes only in the surrounding context of elements (*e.g.*, adding a link to each author, or inserting a "Used Price".) Our modeling should use use "minimal" patterns that only concentrates on elements of interest, and not their surrounding context.

Our Proposal: Visual Relational Modeling. Guided by the dual modeling principles, we develop a novel wrapper framework consisting of a new model and the associated learning and execution techniques. As the key foundation, our propose to construct wrappers with visual features and relational patterns. On one hand, form the Principle of High-level Features: We elevate the level of abstraction for our wrappers to the *visual*-level features of a page—exactly as what human users will see of the page as rendered by a browser, which is probably the highest-level possible. On the other hand, from the Principle of Minimal Patters, we concentrate our patterns to only those *relations* between desired elements (and not surrounding tag sequences). Thus, to see explicitly what "elements" are desired, we require input of *one* example record.

For instance, consider Figure 1.1, supposing we want to extract jobtitle, company, and date. Focusing on these elements, we may describe them as, left(jobtitle, company) (jobtitle is at the left of company) and left(company, location). Note that they hold for both pages of different times.

System Setting: We conclude with concrete definition of our system setting.

Input: *One* or more example data pages, where
one record is labeled with attributes desired.
Output: Wrapper for extracting similar data pages.

3.1 Visual Relational Wrapper Model

At the core of our system, we need a mechanism for specifying a wrapper. For a wrapper W to extract data from a page $\mathcal{P}$, such a specification, or a *model*, should describe what elements on the page are of interest and where they are.

The effectiveness of a wrapper essentially hinges on its model. As the driving mechanism of a wrapper, the model determines the performance of the wrapper and serves as the interface to users who "train" the

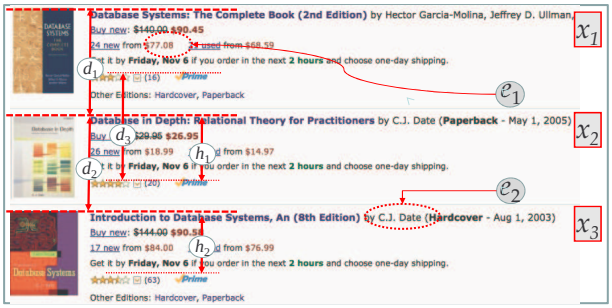

Figure 3.2: Example page fragment (amazon.com).

wrapper. Thus, our requirements (Section 3) for wrapper—accuracy, robustness, and intuitiveness–directly translate into the desired properties for the model.

Thus, we believe that wrapper induction is not simply the problem of learning patterns and inducing a model—the choice of models does matter. As Section 3 explained, while various solutions exist, they all universally assume the "standard" HTML as the representation of their modeling of Web pages. Because their wrapper models similarly amount to the specification of tag *sequence* patterns in *HTML* trees, while their induction approaches differ, they all suffer the limitations inherent in the choice of modeling.

As our main insight, to meet the requirements, our model clearly distinguishes from the traditional specification: We propose *visual relational constraint model* for specifying a wrapper, which elevates page representation to the visual (instead of hidden HTML code) level and minimize the constraints to only relational (instead of sequence) patterns between elements of interest.

Given an HTML data page, which contains a set of data records (which are usually results in response to a query), since a wrapper aims to extract those records, its model must describe, on such a page, how to locate such records—*i.e.*, for each record: *What* are the desired elements? *Where* are them on the page? As our running example, we consider the page fragment, as Figure 3.2 shows.

What: Schema. *First*, what elements are of interest? Essentially, as we are looking for a set of records, we are asking what consists such records, or their "schema." We assume a record as a flat set of attributes, each of which can be omitted or repeated. We found this structure simple yet sufficiently expressive for most data sources. As we focus on extracting *values* of data elements, and not their potential hierarchical structure, we are viewing records as "flattened"–which is nature in most cases. Even for the rare cases when data is nested (*e.g.*, airfare itinerary, where a record contains departure and returning, each can be a record of several attributes, *e.g.*, time and flight), our model can still target the desired elements and extract their values, although without the potential hierarchy (*e.g.*, as time1, flight1, time2, flight2). Further, the flexible multiplicity of attribute occurrence, as we found, is frequently required as data is not always uniform (e.g.,

a book record may not have an *cover*, or may have multiple author).

Thus, as the first component of our model (the "what" component), we define *schema* of a record $(E, \mathcal{T}, \mathcal{Q})$ for specifying a set of attributes $E = \{a_1, \ldots, a_n\}$, their types in $T = \{t_1, \ldots, t_n\}$, and quantifiers $\mathcal{Q} = \{a_1, \ldots, a_n\}$. That is, S specifies some n attributes, each with an attribute name (or attribute identifier) a_i, type t_i, and a quantifier q_i. Comparably, this component can be considered as a set of attributes $E = \{e_i\}$ (represented by attribute names). Each attribute e_i is a 2-tuple $(type, quantifier)$.

Example 1 (Schema): For our example (Figure 3.2), suppose we are interested in, for each book, the cover image or cover, title, author, format (hardcover or paperback), and "Buy New" price. As types, we see that author and format are plain text, title is an *link* (or "anchor text"), cover is an *image* and price is number. As quantifiers, all the attributes will appear exactly once, except author, which may appear multiple times. The schema model of the desired book records is thus E= { cover(*image*, 1), title(*link*, 1), author(*text*, +), format(*text*, 1),newprice(*number*, 1) } ∎

To describe types, the system supports a customizable set of types $\mathcal{T}$, which $e_i : type$ is drawn from, *i.e.*, $e_i : type \in \mathcal{T}$. Even though we keep type set $\mathcal{T}$opened in our framework (for the purpose of customization and flexibility), the implemented type-recognizer in our framework is error-free since $\mathcal{T}$is a generalized concept of standard HTML-tag set. The type set, however, can include any "domain" of values that are of interest to the application and that can be recognized from pages.

To describe the multiplicity of an attribute, *i.e.*, how many values may occur, the system supports the set of quantifiers $\mathcal{Q}$. We adopt the standard regular expression quantifiers, $\mathcal{Q} = \{1, ?, +, *\}$.

Where: Visual Relations. *Second*, where are those elements of interest? While existing wrapper approaches all "address" elements by HTML tag path patterns, we take a fundamentally different view. For describing the "where," as the second component of our model, we provide matching patterns in therms of *constraints* on the elements, where each constraint is gauged at the visual level (and not the HTML tags), and involves only the elements of interest (and not the irrelevant sequence in the surroundings). Each constraint is thus a binary *visual relation* between a pair of desired attributes. Note that in principle, n-ary relations are possible; we choose to use only binary relations, for intuitiveness and simplicity.

Our design of visual relations follows directly from, as Section 3 motivated, the principles of the highest level of presentation and the minimal extent of patterns. To be at the highest level, we gauge the visual perception of users and, to be minimal, we characterize only those desired attributes. Consider Figure 3.2 with the schema in Example 1, how to describe where these attributes are on the page? With visual relations, our patterns would describe how the attributes relate, in terms of visual layout, to each other. For instance, cover is at the *left* of title or left(cover, title); title is at the *top* of price or top(title, price), and cover is at the

left of price or left(cover, title), *etc.*

In determining whether a particular visual relationship holds, we use each element's visual positions as determined by browser rendering—*i.e.*, as human users would see it. Specifically, for a given page, such visual *elements* will be produced by rendering the page as in a browser and then "tokenizing" it into basic units, each associated with visual positions on the page. We characterize each element by its entire span, *i.e.*, , the tight *bounding box* that encloses the element: We view the page as a Cartesian coordinate system, with the top-left corner as the origin (0, 0). On the page, each element is a rectangle with a "start point" (x, y) as its top-left corner, from where each dimension extends a range, *width* and *height* respectively, as a rectangle area, and thus its visual coordinate is (x, y, *width*, *height*). To determine a visual relation of two elements a_1 and a_2, we simply compare their coordinates, *i.e.*, ($a_1.x$, $a_1.y$, $a_1.width$, $a_1.height$) versas ($a_2.x$, $a_2.y$, $a_2.width$, $a_2.height$).

To describe such visual relational constraints in our model, the system should support a set of predicates as the vocabulary. While these predicates may capture various relationships between elements, as Section 3 motivated, we want them to be intuitive and easy to understand by users—and thus we wish to keep these predicates simple yet sufficient in capturing the visual arrangement of records.

What are essential predicates to support? As the essence of visual layouts, we observe that every data page share common presentation characteristics:

- *Two-dimensional* topology: Elements are related to each other in both the x-dimension, *left* and *right*, and the y dimension, *top* and *bottom*. As the relations are symmetric, we support predicates left($\cdot$) and top($\cdot$). *E.g.*, as noticed earlier, in Figure 3.2, we have left(cover, title) and top(title, price).
- *Tabular* alignment: Records are often laid out in some tabular alignment, such as, for the row orientation, horizontally aligned and, for the column orientation, vertically aligned. Thus, correspondingly, we support predicates alignx($\cdot$) and aligny($\cdot$). *E.g.*, in Figure 3.2, since the cover image is vertically aligned with title, their relation aligny(cover, title) holds true.

Overall, to capture these essential characteristics, we need to support only four predicates $\mathcal{V}$ = {left, top, alignx, aligny}. While the choices are naturally motivated by the visual characteristics of record layout patterns, they prove to be very effective in our empirical study (Section 5). While expressive, as only a small number of simple relationships, these predicates are quite intuitive to understand and easy to determine, which indeed meet our requirements.

Definition 1 (Visual Relations): A visual relation between attributes a_1 and a_2 is a binary predicate $r(a_1, a_2)$, where $r \in \mathcal{V} \equiv$ {left, top, alignx, aligny}. Each predicate is determined as follow:

- left(a_1, a_2): true if $a_1.y + a_1.height \leq a_2.x$.
- top(a_1, a_2): true if $a_1.y + a_1.height \leq a_2.y$.
- alignx(a_1, a_2): true if $\neg$left(a_1, a_2) $\wedge$ $\neg$left(a_2, a_1).
- aligny(a_1, a_2): true if $\neg$top(a_1, a_2) $\wedge$ $\neg$top(a_2, a_1). ■

Since a relation describes a predicate between attributes, it is either true or false in *each* record—However, it may not hold uniformly across all records. Some relations may hold for all records, *e.g.*, in Figure 3.2, left(cover, title) does hold for all the records. However, in contrast, for record 1 and 2, observe that title is at the top of format ("hardcover"), which does not hold for record 3 (where title is at the same row as format "paperback"); thus, top(title, format) is inconsistent from record to record. Such "inconsistency" can result from either "client-side" rendering settings or "server-side" data characteristics. Client-side effect comes from the reason that data is longer than the width of its container (*e.g.*, document, browser, etc) and thus automatically goes to a new line. This inconsistency, however, is rather easily to be removed by extending the canvas width in buffer while rendering the page. The technique is very cheap and trivial in implementation. We call the state gained by applying this technique as *unbounded-canvas environment* (will be used in our framework)/.

Therefore, as visual relations may not be consistent across predicates, we need to capture their "fuzziness"—in a probabilistic sense. For our toy example as just mentioned, top(title, format) holds true for 2/3 or 67% of the time, statistically, while left(cover, title) holds 3/3 or 100%. Each visual relation r in our model will thus associate with a probability $p(r)$, written as $r{:}p(r)$, which indicates how likely r will hold true in a record, *e.g.*, top(title, format):0.67 and left(cover, title): 1.0.

Example 2 (Visual Relations): Continuing Example 1, for our example page, what are the visual relations?

Examining every pair of attributes from E, we may identify several visual relations with non-zero probabilities—*i.e.*, holding true in at least one record. For instance, between cover and title, checking each relation r in $\mathcal{V}$, we find that left(cover, title) and aligny(cover, title) hold for all three records, thus both 100% (and top and alignx are of zero probability). For the reversed pair, *i.e.*, (title, cover), only aligny holds (with 100%).

We can similarly check for the remaining pairs, to obtain the set of visual relations $\mathcal{R}$ = {left(cover, title):1.0, aligny(cover, title), aligny(title, cover), top(title, price): 1.0, top(title, format):0.67, left(cover, title):1.0, $\cdots$ } ■

Overall: Wrapper Model. With the schema E and visual relations $\mathcal{R}$ in place, in our system, we

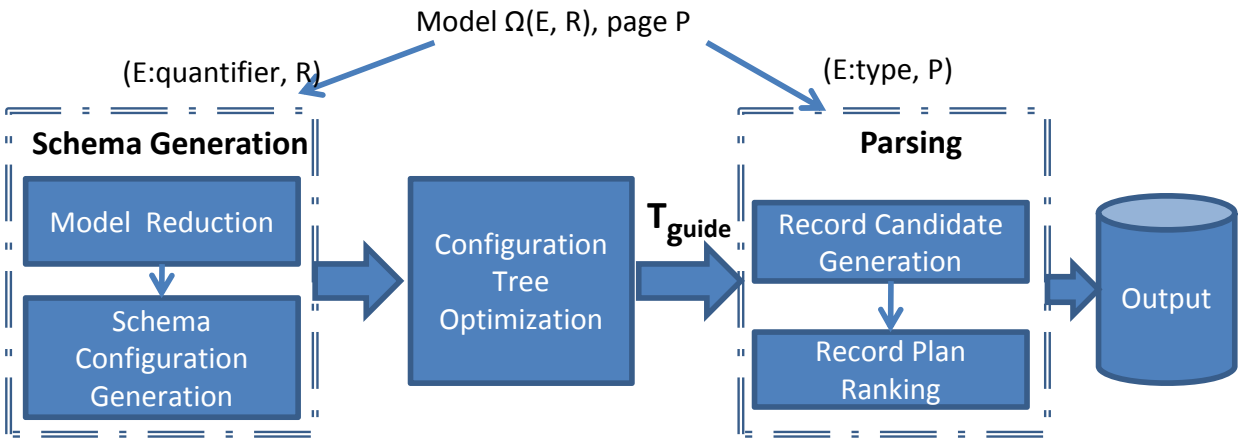

Figure 3.3: Model Execution

define a model $\Omega = (E, \mathcal{R})$, which specifies what attributes and where they are, for a record in our target data page to extract. *E.g.*, for our example (Figure 3.2), Ω consists of the schema in Example 1 and visual constraints in Example 2.

Definition 2 (Visual Relational Wrapper Model): The *visual relational wrapper model* for a data page is a 2-tuple $\Omega = (E, \mathcal{R})$, which specifies the schema and visual characteristics of the records on the page: E is the set of 2-tuple attributes $e(type, quantifier)$ with type $e : type$ and quantifier $e : quantifer$, and $\mathcal{R}$ the set of visual relations between the attributes. ■

3.2 Model Execution: Extracting Data

In this section, we formulate the model execution architecture. Given a model Ω=$\{E, \mathcal{R}\}$ and a page $\mathcal{P}$, we need to output a maximal set of non-overlapping tuples (*i.e.*, data records) $\Upsilon = \{\Upsilon_i\} \in \mathcal{P}$ which is generated by Ω. We call the probability that a tuple Υ_i is generated by visual model Ω is $p(\Upsilon_i|\Omega)$. If $p(\Upsilon_i|\Omega)$ is too small, it is unlikely that Υ is generated by Ω and thus not a good candidate tuple to be extracted. Therefore, we use a *generative threshold* θ_0 as lower-bound of generative probability to determine if a candidate tuple Υ_i is considered to be generated by Ω. In other words, a candidate tuple Υ_i is a *valid tuple* if and only if $p(\Upsilon_i|\Omega) \geq \theta_0$. The higher $p(\Upsilon_i|\Omega)$, the better tuple Υ_i is. $P(\Upsilon_i|\Omega)$, hence, also indicates the ranking score of a candidate tuple. Consequently, the output of our model extraction is a maximal non-overlapping set of *valid tuples* $\{\Upsilon_i\}$ with highest ranking score (Equation 3.1)

$$\Upsilon = Argmax_{\{\Upsilon_i|p(\Upsilon_i|\Omega)\geq\theta_0\}} \sum_{\Upsilon_i\in\{\Upsilon_i\}} p(\Upsilon_i|\Omega) \tag{3.1}$$

Note that visual model Ω, by definition, holds the statistical measures of visual relations among attributes of a data record. Each of such measures, in fact, represents a generative distribution of one relation between two attributes. For example, with a simple pair of two 1-quantifier attributes e_i, $e_j \in E$ ($e_i : quantifer = 1$ and $e_j : quantifer = 1$), relation $r(e_i, e_j) : p_r$ has only two possible *instantiations*: $r(e_i, e_j) = 1$ or $r(e_i, e_j) = 0$ (*i.e.*, r holds or not hold) with probability of p_r and $(1 - p_r)$ respectively. The real distribution, however, can be much more complicated (Section 3.2.1) since we support all possible quantifiers. Each combination of $|R|$ relation instantiations, in turn, denotes a specific alignment layout of target data records which we call *relational schema configuration* (or *schema configuration* in short). Since schema configurations capture all possible variations of alignment layout of a data record, a record candidate essentially follows one specific configuration. Our extraction framework is, thus, three-phased. *First*, consider visual model Ω as a visual alignment generative model, we generate schema configurations and their generative probabilities (Section 3.2.1). *Second*, toward an efficient parsing, we optimize the parsing order in order to identify invalid configuration as soon as possible, the information are stored inside a tree structure called configuration tree T_{guide} (Section 3.2.2). *Third*, we parse page $\mathcal{P}$followed the guidance of T_{guide} and aim for the top-ranked dataset which satisfies Equation 3.1 (Section 3.2.3).

3.2.1 Relational Schema Generative Model

As Section 3.1 discussed, our visual model Ω captures the relative alignment information between each pair of two attributes (*i.e.*, visual relations). As such, two data records should be considered the same (*i.e.*, identical generative probability) w.r.t. generative behavior from Ω as long as they share the same schema configuration. Implicitly holding statistical distributions of visual relations, our visual model, thus, is a generative model of schema configuration. The generative probability of a record implies generative probability of its schema configuration. This section explains internal components of the schema configuration generation.

Model Reduction

Schema configuration is a combination of relation instantiations. Ideally, each relation $r(e_i, e_j)$ of two attributes a_i, a_j should only contains two instantiations: either *hold* or *not-hold*. Unfortunately, this is not always the case. A multi-instance attribute (*e.g.*, author in Amazon's books) with "+"/"*" quantifier can make its relation become *fuzzy* since the relation might hold with some instances but not-hold with the others. Such fuzziness is further deepened with optional attributes (*i.e.*, "*" and "?"). Identified the source of relation instantiation fuzziness, we therefore want to reduce the quantifier set. Firstly, we observe that

$(e^+) = (e^1)(e^*)$ and thus a "+"-attribute can be replaced by one "1"-attribute and one "*"-attribute. This conversion is done by quantifier decomposition operator $\mathcal{Q}_D$ (Definition 3). Secondly, we further observe that an optional attribute become non-optional if we include *null* in the data type. This transformation (denoted by $\mathcal{Q}_R$) is formalized in Definition 4.

Definition 3 (Quantifier Decomposition): A quantifier decomposition operator ($\mathcal{Q}_D$) is an operator which transforms a visual model $\Omega = (E = e_1, \ldots, e_m, \mathcal{R})$ containing some "+"-quantifier attribute e_k into model $\ddot{\Omega}$=($\ddot{E}$, $\ddot{\mathcal{R}}$) without such attribute by replacing $e_k(type, +)$ by two attributes $e_k^1(type, 1)$ and $e_k^*(type, *$ so that

- $\ddot{E} = \{e_1, \ldots, \mathbf{e}_k^1, \mathbf{e}_k^*, e_{k+1}, \ldots\}$
- $\ddot{\mathcal{R}} = \mathcal{R} - \mathcal{R}_k + Replace(\mathcal{R}_k, e_k, e_k^1) + Replace(\mathcal{R}_k, e_k, e_k^*)$ where $\mathcal{R}_k$ is relation set of e_k ∎

Definition 4 (Optional Removal): An optional removal operator ($\mathcal{Q}_R$) is an operator which transforms any optional attribute $e_k(type, quantifier)$ of visual model Ω into non-optional attribute $\ddot{e_k}(type \cup null, quantifier)$ where $\ddot{e_k} : quantifier$ ="1" if $e_k : quantifier$ = "?" or $\ddot{e_k}$:$quantifier$="+" if e_k:$quantifier$="*".

By applying two operators $\mathcal{Q}_D$ and $\mathcal{Q}_R$ in that order, the induced model guarantees to have only two types of quantifier: "1" and "+". This 2-step model transformation seems to pose internal conflict (*i.e.*, first remove +-attributes and later transform to +-attributes again) but, in fact, it does not. After 2-step transformation, every +-attribute is guaranteed to have type with *null* included. This plays a crucial role to identify the hidden distribution of relation instantiation which decides the generative behavior of Ω. From now on, we assume visual model contains only quantifier "1" and "+".

Relational Schema Configuration Generation

We can safely assume that every +-attribute contains at most N_{max}^* instances. N_{max}^* is called *instance-bound*. Empirically, in our system which operates on dataset 2Y5D, we choose N_{max}^*=3. From model reduction, we know that every +-attribute of model Ω (after reduced) accept *null* as a valid type. As a sequence, a +-attribute e_i is comparable with a N_{max}^*-tuple $\{e_i^1, \ldots, e_i^{N_{max}^*}\}$ where e_i^k can be a *null* instance.

Relation instantiation: implicit distribution With the probabilistic relation set $\mathcal{R}$, we now define the underlying distribution of each relation $r(e_i, e_j) \in \mathcal{R}$. As noted above, relation instantiation depends entirely on relevant attribute's quantifiers. As such, given p_r as the probability of relation $r \in \mathcal{R}$, we have three scenarios of set $\{e_i$:$quantifer$, e_j:$quantifier\}$ as follows.

First - { *"1", "1"*}: There are two possible instantiations $Inst_1$ when $r(e_i, e_j)$ *hold* and $Inst_0$ when $r(e_i, e_j)$ *not-hold* with probabilities

$$P_1(r = Inst_k|\Omega) = \begin{cases} p_r & if\ k = 1 \\ 1 - p_r & if\ k = 0 \end{cases} \tag{3.2}$$

Second - { *"1", "+"*}: Without loss of generality, we assume e_j is the "+"-attribute. Thus, relation r is actually a set of N^*_{max} *primitive relations* $r(e_i, e^k_j)$ with k=1 ... N^*_{max}. Intuitively, r has $(1 + N^*_{max})$ instantiations $\{Inst_k\}$ where $Inst_k$ indicates that there are exactly k primitive relations hold. There are $C_k^{N^*_{max}} = \dfrac{n!}{k!(n-k)!}$ different picks for such k-set of *hold* relations from N^*_{max} primitive relations; each with probability of $p^k_r.(1 - p_r)^{N^*_{max}-k}$. Therefore, the probability of a relation instantiation $Inst_k$ is:

$$P_2(r = Inst_k|\Omega) = C_k^{N^*_{max}}.p^k_r.(1 - p_r)^{N^*_{max}-k} \tag{3.3}$$

Third - { *"+", "+"*}: Similarly, this relation is actually a set of $(N^*_{max})^2$ primitive relations $r(e^u_i, e^v_j)$ with u, v=1 ... N^*_{max}. Thus, r has $(1 + (N^*_{max})^2)$ instantiations $\{Inst_k\}$ where $Inst_k$ indicates that there are exactly k primitive relations hold. The probability of a relation instantiation $Inst_k$ is:

$$P_3(r = Inst_k|\Omega) = C_k^{(N^*_{max})^2}.p^k_r.(1 - p_r)^{(N^*_{max})^2-k} \tag{3.4}$$

Generation Behavior and Generative Probability We now discuss how model Ω generates relational schema configurations. By definition, model Ω represents $n_R = |\mathcal{R}|$ distributions of visual relation. For each relation $r \in \mathcal{R}$, Ω simply decides to select one instantiation $Inst^r$ with probability $P(Inst^r|\Omega)$. The final result of n_R such selections on all $r \in \mathcal{R}$ is an n_R-set of relation instantiations which we call *schema configuration*. The probability that Ω generates a configuration is called *configuration generative probability*. We now formalize such probability. Assume all relations in $\mathcal{R}$ are mutually-independent then each selection of relation instantiation is also independent from others. As such, configuration generative probability $P(\{Inst^r\}|\Omega)$ of a configuration that relation r has instantiation $Inst^r$ is product of its instantiation probabilities $P(r = Inst^r|\Omega)$ (Equation 3.5). A configuration with generative probability not less than generative threshold θ_0 is considered a *valid configuration*. Ones with probability less than θ_0 are called *invalid configuration*.

$$P(\{Inst^r\}|\Omega) = \prod_{r \in \mathcal{R}} P(r = Inst^r|\Omega) \tag{3.5}$$

Where $P(r(e_i, e_j) = Inst^r|\Omega) =$

$$\begin{cases} P_1(r = Inst^r|\Omega) \text{ if both } e_i, e_j \text{ are 1-attributes} \\ P_2(r = Inst^r|\Omega) \text{ if either } e_i \text{ or } e_j \text{ is 1-attribute} \\ P_3(r = Inst^r|\Omega) \text{ if neither } e_i, e_j \text{ is 1-attribute} \end{cases}$$

3.2.2 Configuration Tree: Parsing Efficiency

Invalid configurations are unimportant in our extraction framework since they represent data records which are unlikely to be generated from Ω. Generally, to identify if a configuration $C = \{Inst^r_C\}$ is invalid ($Inst^r_C$ is instantiation of r in C), we need to check its generative probability follows Equation 3.5. Intuitively, if there exist a subset $C_{sub} \in C$ (called *partial configuration* of C) so that $\prod_{Inst^r_C \in C_{sub}} P(r = Inst^r_C|\Omega) < \theta_0$, then C is definitely an invalid configuration (since $\prod_{Inst^r_C \in C} P(r = Inst^r_{k_r}|\Omega) \leq \prod_{Inst^r_C \in C_{sub}} P(r = Inst^r_C|\Omega)$). Such C_{sub} is called *invalid partial configuration*. Consequently, an invalid configuration can be identified without the need to identify all of its relation instantiations as long as we find an invalid partial configuration of it. To capture the generative probability of such partial configurations, we need to consider the configuration generation process as a sequence of relation instantiation generation. The generation process, with respect to a specific generative sequence $(r_1, r_2, ..., r_{n_R})$, can be represented by a n_R-depth tree called *configuration tree*. A node in level i represents a partial configuration $(Inst^{r_1} \dots Inst^{r_i})$, each node in level i has exactly $N^{r_{i+1}}_{Inst}$ children with $N^{r_{i+1}}_{Inst}$ is the number of instantiations of relation r_{i+1} . Each child in level $(i + 1)$ is a partial configuration which extends from its parent configuration with one specific instantiation of r_{i+} (denoted by the edge from its parent). In general, level i of a configuration tree w.r.t order $(r_1 \dots r_{n_R})$ holds all possible partial configurations of a set of relation $r_1 \dots r_i$. Therefore, Leaf nodes are schema configurations (*i.e.*, partial configuration of all relations) with configuration generative probability. The sequence $(r_1, r_2, ..., r_{n_R})$ is called *parsing order*.

Example 3 (Configuration Tree): Assume model $\Omega = (E, \mathcal{R})$ from Amazon.com has E={title^1, author$^+$, UsedPrice^1 } where superscript denotes attribute's quantifier. $\mathcal{R}$ = {r_1 = left(title, UsedPrice):0.7, r_2 = left(author, UsedPrice):0.6, r_3 = top(title, UsedPrice):1}. Generative Threshold θ_0=0.1, instance bound N^*_{max}=2 for book on Amazon. Notationally, we write $r(I_k : p)$ to indicate instantiation $Inst_k$ (*i.e.*, there are exactly k primitive relations hold) of relation r has probability of p. As such, we have three distributions $r_1(I_1 : 0.7, I_0 : 0.3)$, $r_2(I_2 : 0.6^2, I_1 : 0.24, I_0 : 0.4^2)$, $r_3(I_1 : 1, I_0 : 0)$. Figure 3.4-a, shows the configuration generation w.r.t relation order $r_1 - r_2 - r_3$. The tree is generated as follows: First, start from root (level 0), consider to first relation in parsing order (*i.e.*, r_1), then this relation has two instantiation $I^{r_1}_1$:07 = hold and $I^{r_1}_0$:0.3 = not-hold. As such, we have two branches from root indicate these two instantiation of r_1 with

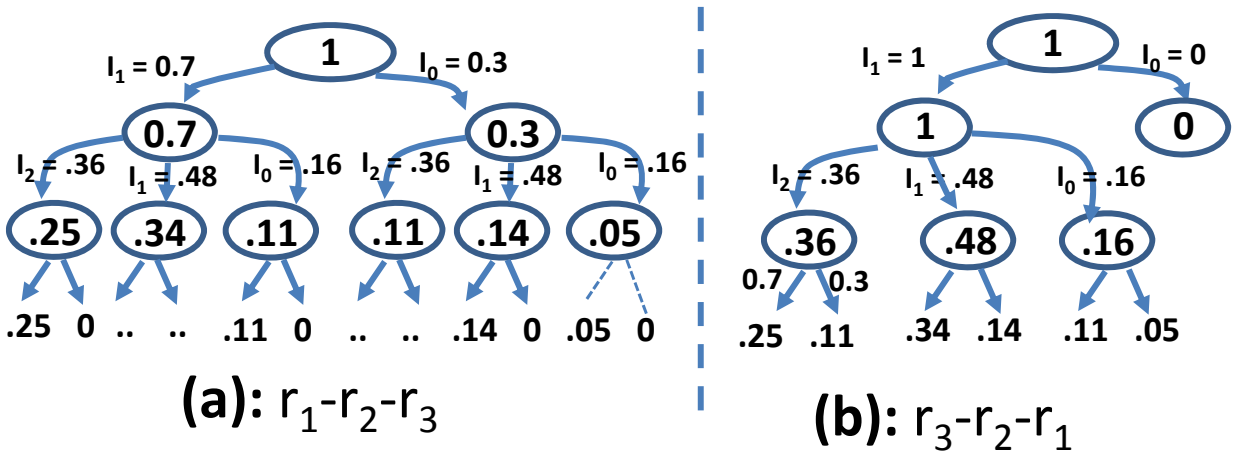

Figure 3.4: Configuration Tree Generation

probability 0.7 and 0.3 respectively. The two child nodes on level 1 are, therefore, two partial configurations $\{r_1 = I_1^{r_1}\}$ and $\{r_1 = I_0^{r_1}\}$. Each of these two nodes generates three children in level 2 since relation r_2 has three different instantiations, etc.

Paring Order - Toward Efficient Parsing Observation on Figure 3.4-a shows that even more than half of the generated configurations are invalid (*i.e.*, 7 out of 12), most of them (*i.e.*, 5) can only be identified when the tree is fully generated. With different parsing order, we observe a major difference on configuration tree in Figure 3.4-b. All invalid configurations except one can be identified without the need to generate to full configuration. One configuration represented many record candidates. Configuration tree pruning, therefore, is a crucial step toward an efficient parsing.

As the above observation motivates, essentially, we need to identify the parsing order which leads to the best pruned configuration tree (*i.e.*, smallest number of nodes). This problem shares some similarity with decision tree classification problem where we need to identify the best attribute that maximizes classification capability first. In our context, the best relation is the one it can lead to invalid configuration as soon as possible. As a result, comparable with several heuristics used in Decision Tree Classification, we can apply a simple heuristic by picking relation which contain the lowest instantiation probability p_{min}. For example, in Example 3, we favor r_3 first since $p_{min}(r_3) = 0$ and r_1 last since $p_{min}(r_1) = 0.5$.

In our implementation, however, we decided to take brute force approach to find the best parsing order because of the following reasons. Firstly, the parsing order is model-dependent only and thus it can be done offline once and used in every extraction pages. Secondly, the number of parsing orders is quite small (*e.g.*, 24 for 4-attribute model) and generating a tree is extremely fast (because all distributions of relation instantiation are known) so brute force approach is actually fast. Lastly, saving one branch of the pruned tree

means a huge save in the parsing phase since there are many data record candidates match that instantiation branch. The algorithm is, thus, straightforward, for each parsing order, from root node we expand next level nodes by instantiations of the first relations. A new node is then expanded again by instantiations of the next relation as long as its probability $\geq \theta_0$. Finally, after the tree is generated, any leaf node either not in depth-n_R or has generative probability less than θ_0 is removed along with its edges. The number of remaining nodes determines the size of configuration tree with that parsing order. Output the smallest tree.

3.2.3 Parsing

This section presents the parsing framework follow a pruned configuration tree T_{guide}. We first generate attribute candidates from page $\mathcal{P}$, then prune them using distance-based clustering. Candidates of different attributes are then combined together w.r.t parsing order in configuration tree to form valid data records. Ranking will be applied on non-overlapped sets of valid records to determine the best output dataset.

Attribute Candidate Generation

This section introduces the technique to generate and shorten the set of attribute candidates from a page $\mathcal{P}$for a given model $\Omega = (E=\{e_1, \ldots, e_n\}, \mathcal{R})$. Basically, for each attribute $e \in E$, our type-recognizer generates a list of data elements which match e : *type*. This list, however, can be large if e : *type* is too general. This fact motivates us to develop a method to shorten the number of candidate for each attribute.

Visual Regularity: Record regularity has been used by several extraction methods such as tree-alignment or pattern-based approaches. These approaches, however, only try to utilize the regularity in HTML source code level which results in severe limitation in many types of web pages. The scenario of yahoo hotjob in Figure 1.1-b illustrates this limitation. We, therefore, want to leverage the regularity abstraction to visual layer to overcome the aforementioned limitation. In Figure 1.1-b, even the format of even and odd data records is different, the vertical distance between the same attribute of two consecutive records are constant (*approximately*).

Definition 5 (vertical distance): Let d_i=< x_i, y_i, w_i, h_i >, d_j=< x_j, y_j, w_j, h_j > be two data elements with their rendering positions top-left (x, y), width w and height h. A vertical distance between d_i and d_j is Γ (d_i , d_j)=|x_i - x_j|. ∎

Definition 6 (Γ-cluster): A ordered list of data elements D=$\{d_1, \ldots, d_m\}$ ($m \geq 3$) forms a Γ-cluster if and only if any pair of two consecutive elements (d_k, d_{k+1}) ($k \in [1, m\text{-}1]$) has the same vertical distance $\Gamma(d_k, d_{k+1})= \Gamma$. Γ is called **step** of the cluster.

Claim 1 (Visual Conservation): Let Υ_i, Υ_j, Υ_k is 3 consecutive n-tuples which are generated from visual model $\Omega = (E=\{e_1, \ldots, e_n\}, R)$ where $\Upsilon_t = \{d_{t1}, d_{t2}, \ldots, d_{tn}\}$ with $(t=i/j/k)$, then the following properties hold for any p_1 ,$p_2 \in [1,m]$ in unbounded-canvas environment:

1. Internal conservation: $\Gamma\ (d_{ip_1}\ ,d_{ip_2}\ )=\Gamma\ (d_{jp_1}\ ,d_{jp_2})=\Gamma\ (d_{kp_1}\ ,d_{kp_2})$
2. External conservation: $\Gamma\ (d_{ip_1}\ ,d_{jp_1}\ )=\Gamma\ (d_{jp_1}\ ,d_{kp_1}\ )=\Gamma\ (d_{ip_2}\ ,d_{jp_2}\ )=\Gamma\ (d_{jp_2}\ ,d_{kp_2})$

Interestingly, from external conservation characteristic (in unbounded-canvas environment), we also have $\Gamma(e_{ki}, e_{(k+1)i}) = \Gamma(e_{kj}, e_{(k+1)j})$ with $k \in$[1, n] and $i, j \in$[1, m] which leads to Claim 2.

Claim 2 (Preserved Attribute Cluster): Assume a parsing page has n data records generated from visual model $\Omega = (E=\{e_1, \ldots, e_m\}, \mathcal{R})$ (*i.e.*, n extracted m-tuples) $\Upsilon_k = \{e_{k1}, e_{k2},\ldots, e_{km}\}$ $(k=1,..,n)$, then the following statement holds: if $D_i = \{e_{1i}, \ldots, e_{ni}\}$ is a Γ-cluster of attribute e_i then $D_j = \{e_{1j}, \ldots, e_{nj}\}$ is also a Γ-cluster of attribute e_j(with any pair of attributes e_i,$e_j \in E$)

This claim leads to the algorithm to filter out candidate sets of attributes in visual model. Because the claim infers that all the candidate sets for all attributes in visual model must be clusters with the same vertical distance. This algorithm is just one part of the framework, and due to space limitation, we only describe the main idea of the algorithm. We first try to build clusters for each candidate. Second, we compare steps from clusters of different attributes. An attribute cluster is kept if for each other attribute, we can find at least one cluster with the same step.

Example 4 (link cluster): : In Amazon example in Figure 3.5, consider a data record has only two elements title and price. Therefore, visual model $\Omega = (E, \mathcal{R})$ has E=$\{$title, price$\}$ and E:*type*=$\{$*link*, *number*$\}$. Obviously, the initial candidates for title are all links on the page. We have some Γ-cluster such as **{menu link}, {title}, {buy new}, {Used & new}**, the first one is a d-cluster while the others are D-cluster. Clearly, there is no d-luster on price candidate set (*i.e.*, type *number*). This means only D-clusters are kept for both candidate sets. Elements of **{menu link}** are no longer candidates for title. ∎

Valid Record Generation

A record candidate (n_R-tuple with $n_R = |\mathcal{R}|$) is simply any combination of attribute candidates with respect to attribute quantifier $\Upsilon_i = (c_1, c_2 \ldots c_m)$ where c_k is a set of candidates for attribute e_k. c_k is either 1-set/ N^*_{max}-set if e_k is "1"-attribute/ "+"-attribute respectively. Number of such record candidates is huge but only a portion of them are valid records which belong to some *valid configuration*. Our configuration tree is a perfect structure to determine how to parse a candidate (*i.e.*, check its relation instantiations) in an efficient manner so that we can eliminate invalid candidates without the need to check all of its relations. In

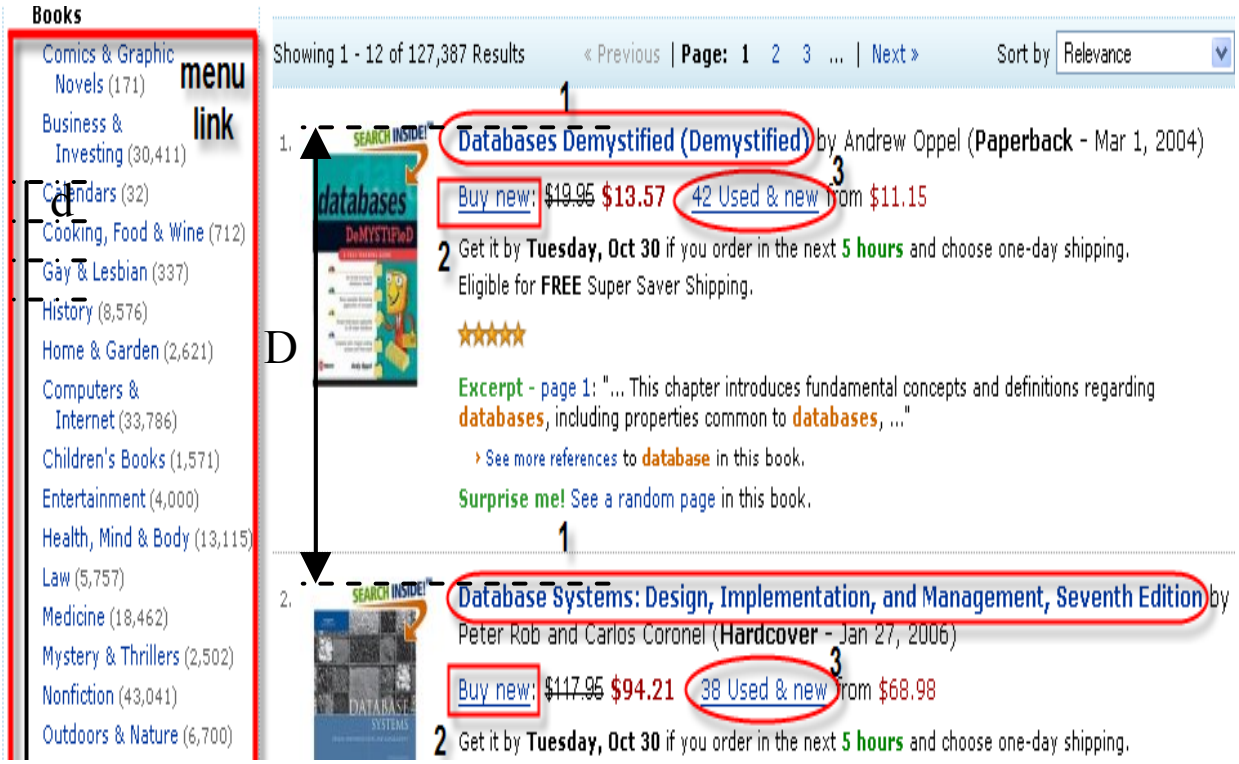

Figure 3.5: Reduce Candidate Set by Distance-based Clustering

a different view, if we gradually expand record candidates follow the structure of T_{guide}, we will finally reach all valid records and avoid invalid ones. With that principle in mind, we generate a *valid record tree* T_{valid} with the same structure as T_{guide}. The only different is the content of each node. Each node of T_{valid} keeps a set of partial record candidates which satisfy the configuration path to it (*i.e.*, satisfy all of the relation instantiations along the path). Start from root with empty partial tuple set. From a node level k (which contain several partial tuple t_k), for each branch $r(e_i, e_j) = Inst^r$ from this node, we generate partial tuple set of the node in level $(k+1)$ as follows: *First*, if two attribute e_i and e_j are already covered in tuple t_k then this tuple is kept in node $(k+1)$ if it satisfies $r(c_i, c_j) = Inst^r$ and removed otherwise. The partial subset retrieved in $k+1$ node, in this case, is a subset of the set in node k. *Second*, if any of attribute e_i or e_j is not covered in t_k (or both) then we find candidate for that attribute (c_i for e_i and/or c_j for e_j) from the attribute candidate set so that $r(c_i, c_j) = Inst^r$, the new partial candidate gained by adding this attribute candidate into r_k will be put into the set of node level (k+1). Repeat this step from root to all leaves. This generation process, guarantee we only generate tuples with valid configurations. Invalid ones have been pruned on-the-fly since their configurations have been pruned from T_{guide}. Tuples in leaf nodes of T_{valid} are all valid records we want to find.

Null relation: The basic operation in valid record tree generation above is to determine instantiation

of a primitive relation between two attribute candidates. Since attribute candidate has specific position, the decision is straightforward except for *null* candidate. Primitive relations of a null candidate are call *null relations*. The question is should a null relation be considered as *hold* or *not-hold*? As long as the attribute accepts null, *null* is considered as "good" candidate. Since *r:p* indicates the probability that *r* hold (and not-hold with probability 1-*r:p*), a good candidate should follow the instantiation with greater probability. As such, since we consider *null* is good, a primitive null relation *r* is considered as *hold* if $P(r = \text{"}hold''|\Omega) \geq 0.5$ and *not-hold* otherwise. We, however, still want to avoid the *null-syndrome* which indicates the choice of null for every possible optional attribute. While there is nothing wrong in principle, we prefer to have a not-null candidate if they share the same configuration. We guarantee this by always chose a non-null attribute candidate before null-candidate in Plan Ranking.

Plan Ranking and Final Output

Two valid tuples Υ_1=(e_{11},e_{21}, ..., e_{m1}) and Υ_2=(e_{12}, e_{22}, ..., e_{m2}) are considered as *non-overlapping* tuples if and only if $e_{1k} \cap e_{2k} = \emptyset$ with any $k \in [1...m]$. A plan is a decision of picking a set of non-overlapping valid tuples Υ=$\{\Upsilon_1, \Upsilon_2, ..., \Upsilon_t\}$ as output dataset. Our ultimate goal is to find the best output plan rather than just find the best output tuple. We call *plan-rank* is a measure to determine which plan is better output. Since each output have a specific ranking score (*i.e.*, its configuration's generative probability), it is natural to consider a accumulative plan-rank which is sum of all ranking scores of a plan's tuples Υ_i in plan. The best output plan is the one with highest plan-rank (Equation 3.1). Intuitively, as long as a valid tuple can be added into an existing plan (*i.e.*, no overlapping), the new plan is always better than the old one because $P(\Upsilon_k|\Omega) \geq \theta_0 > 0$ with any valid tuple Υ_k. As a result, in the context of our paper, a plan always refers to a maximal plan (*i.e.*, the plan that can not be expanded). Basically, there are two approaches to select output plan: *top-k ranking* and *greedy*. Due to paper's space limitation, we only describe their main ideas (straightforward for implementation).

Top-k ranking: in this approach, we first generate all maximal plans from the set of valid tuple acquired in Step 1. Second, we rank all generated plans by their plan-ranks. Last, we will output top-k plan from the ranked list. In our extraction scenario, we only consider top-1 plan which is the best output. This approach is obviously inefficient. It only works acceptably if we have a good generative threshold θ_0 which guarantees to output a small number of valid tuples.

Greedy matching: As indicated above, top-k ranking approach gives us the best output. That approach, however, is not good in term of efficiency. We propose greedy approach to remove the overhead of generating all maximal plans in top-k approach. The basic idea of this approach is to *push* all valid tuples

(got from above step) into a stack in descending order of their configuration probability (*i.e.*, push tuples with lower configuration generative probability first). Start from an empty plan, we expand the plan by *pop* out the the first valid tuple which does not overlap with any of the tuples in the current plan. The greedy approach stops whenever the stack is empty.

Note: Experimental result of this paper is conducted from Greedy Matching approach which proves to be very efficient and accurate.

Chapter 4

Model Induction

Section 3.2 explains how to extract data records from web pages given a visual model $\Omega(E, \mathcal{R})$ and a generative threshold θ_0. The question, consequently, is how to have a good estimation of those two concepts from one training example - which is the input of our system. As Chapter 3.1 mentions, Ω captures relation fuzziness of data records which results from either client side or server side inconsistency. That means visual model, in fact, holds structure estimation of one data record on a specific source (*i.e.*, intra-source structure). A training example is just one concrete instance thus cannot capture such fuzziness. An optimal visual visual model, therefore, essentially derives from a good sample dataset (which have approximate distribution with the source dataset). Assume our training page $\mathcal{P}$is a representative page which holds such sample dataset. The model Induction framework thus contain two phases : 1. *Extraction*: extracting sample dataset Υ from page $\mathcal{P}$; 2. *Estimation*: estimate Ω Υ then estimate θ_0 from Ω and Υ.

4.1 Extraction

Given an unidentified model and threshold, extraction algorithm in Chapter 3.2 is thus inapplicable. We, however, have one training example labeled on page $\mathcal{P}$. As a result, all possible features and/or characteristics of this example need to be exploited in order to ensure a good and complete output. Assume page $\mathcal{P}$contains n m-tuple data records $\{\Upsilon_i = (e_{i1}, \ldots, e_{im})\}$, $(i = [1, n])$ on training page $\mathcal{P}$. Where, e_{ik} is a data element corresponding with attribute e_k of data record Υ_i. Let Υ_0 be the labeled training example.

On the view of each training attribute element e_{0k}: from this attribute element, the type-recognizer reveals data type of attribute e_k. Besides, e_{0k} also reveals a partial information for e_k : $quantifier$ (*i.e.*. + if users label several instances and 1 if users label one instance). The partial quantifier, however, lacks optional characteristic which we will recover later (*i.e.*, + can be * and 1 can be ?). More importantly, e_{0k} gives us clue of how to get to the correct clusters for each attribute e_k from the clustering algorithm. Basically, for all of the clusters retrieved from the general clustering algorithm, we first identify the ones which contain labeled attributes (*seeded clusters*). From seeded clusters, we can easily identify the desired

data region (which is the parent HTML tag of all seeded clusters). Finally, we chose all clusters belongs to this region and have the same step with seeded clusters (since optional attributes breaks up original clusters into several sub-clusters of the same step). Candidate for each attribute e_k is a set of clusters type $e_k : type$ from those clusters. Two attributes of the same type thus share the same set of clusters. Those clusters are called *clean* clusters. Basically, a clean cluster of attribute e_k is either a correct cluster of this attribute or a cluster of other attribute with the same data type.

On the view of visual relations between training attribute elements. Each visual relation has some specific instantiation on training example. To exploit that information, we need to identify a relation or set of relations which must share the same instantiations over any data record Υ_i. Those relations, if exist, must be implied from the HTML source code (thus they hold for every data records). Theorem 1 defines such relations.

Theorem 1 (Visual Invariant): Given a deep-web source dataset $\Upsilon = \{Upsilon_k\}$ where Υ_k is a m-tuple data record with m attributes $a_1, \ldots, a_m$. If any of the following sets of visual relations hold for a data record $\Upsilon_k \in \Upsilon$ in unbounded-canvas environment, it must also hold for every other data record $\Upsilon_h \in \Upsilon$ in that environment.

- top(a_i, a_j)
- aligny(a_i, a_j) and left(a_i, a_j)

proof: From the meaning of unbounded canvas environment, if we have top(a_i, a_j) holds true for Υ_k, there must be some explicit delimiter in HTML source code (*i.e.*, ,
 or <tr>) which separates the two attributes of Υ_k. Those of other records Υ_h are also populated from server database and thus essentially subject to the same delimiter of the HTML source which means top(a_i, a_j) holds on them. On the other hand, If aligny(a_i, a_j) is true for Υ_k, it implies that a_i and a_j are populated from server database without any line delimiter in between. In that case, left(a_i, a_j) guarantees that a_i is populated first, then a_j. This should also be the same for every other data records and we have both aligny(a_i, a_j) and left(a_i, a_j) hold on any Υ_h.

False candidates: What? Since all record candidates are generated from *clean* clusters, a wrong one must has at least one incorrect attribute which, either: (**1**) Be *null* while in fact it is not (*i.e.*, *immature candidate*) or (**2**) Comes from other data records (*i.e.*, *false inter-record candidate*) or (**3**) Comes from the correct date record but in wrong order (*i.e.*, *false intra-record candidate*).

Immature candidate: This type of candidate, intuitively, results in a reduction in the total number of attributes of output dataset Υ comparing to the output plan which replaces immature candidate by the corresponding *mature* data record.

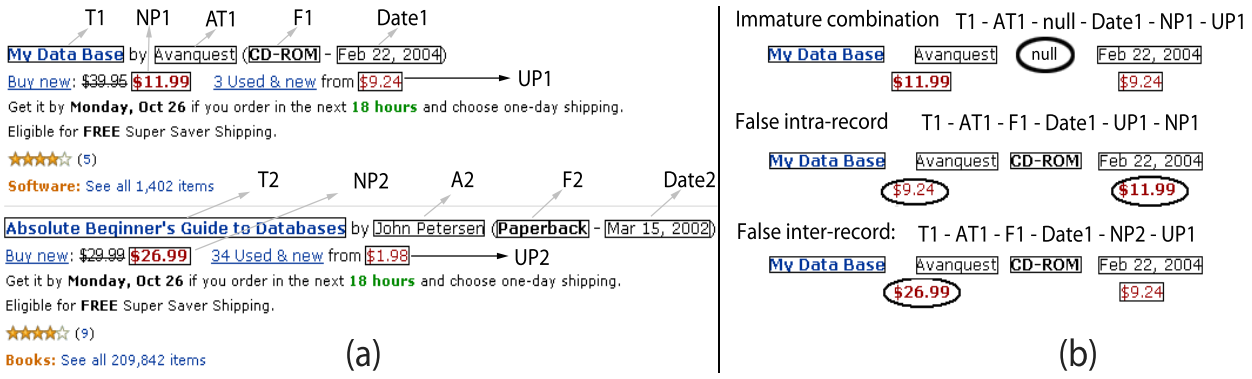

Figure 4.1: Types of false combination from clean clusters

False intra-record candidate: This type of candidate occurs only if there are at least two attributes of the same data type. In our case, we consider two attributes are non-overlapped. Therefore, either one stays on top of the other or they are horizontally aligned and one stays on the left of the other. In either case, the same visual constraints must be held on every correct data record as the Visual Invariant theorem (*i.e.*, Theorem 1) states. Since this type of candidate breaks the Visual Invariant comparing to training example, we can avoid it by maintaining this visual invariant in candidate generation.

False inter-record candidate: This wrong combination occurs when attributes of two or more data records are considered as one candidate. Since output data records are non-overlapped, correct attributes in the region of this false candidate (*e.g.*, *NP1, T2, A2, F2, Date2* in Figure 4.1) cannot be considered in other output candidates and thus be missing from the output dataset. As a result, this type of combination also results in a reduction in the total number of output attributes.

Characteristics of false candidates mentioned above lead us to a modification of the ranking equation (*i.e.*, Equation 3.1) in Extraction Framework. Rather than maximizing the generative probability of output plan (*i.e.*, output dataset) which is not available in this induction phase, we aim for maximizing the total number of output attributes while still maintaining the visual invariant in each record candidate. Equation 4.1 shows this modification in which every Υ_i is non-overlapped with others and satisfies the Visual Invariant proposed by the training data record Υ_0. $|\Upsilon_i|$ stands for the size record candidate Υ_i (*i.e.*, number of attributes of Υ_i).

$$\Upsilon = Argmax_{\{\Upsilon_i | Visual\_Invariant(\Upsilon_i)=true\}} \sum_i |\Upsilon_i| \tag{4.1}$$

Following all the principles above, Algorithm 1 shows the pseudo code for extracting output data records from training example $\Upsilon_0 = (e_{01}, e_{02}, \ldots, e_{0m})$ and training page $\mathcal{P}$. Input of this algorithm is clean clusters

retrieved from Clustering algorithm with seed. For the purpose of efficiency, we greedily expand a data record to as many attributes as possible yet still guarantee not to violate the Visual Invariant (line 8 to 20). This avoids including immature candidates in the final output dataset. We also incorporate *Internal Conservation* condition (*i.e.*, Claim 1.1) into candidate generation process (*i.e.*, line 10) to reduce the number of tested dataset. The optimal output dataset is the one with maximum number of attributes (*i.e.*, line 36).

4.2 Estimation on Extracted Sample Dataset

Visual Model Induction With the sample dataset extracted, estimating visual model $\Omega = (E, \mathcal{R})$ is straightforward. As mentioned above, while $E : type$ is determined, the quantifier set still lacks of optional characteristic. From extracted dataset Υ, we add optional characteristic for attribute e_i (*i.e.*, changing + to * and 1 to ?) if there is at least one record $\Upsilon_k \in \Upsilon$ in which e_i is missing. For each pair of attributes (a_i, a_j), we generate every relations (top, left, alignx, aligny) with probability following Equation 4.2.

$$r^*(a_i, a_j) : p = \frac{\mid \{\Upsilon_k \in \Upsilon \mid r(e_{ki}, e_{kj}) = 1\} \mid}{\mid \Upsilon \mid} \tag{4.2}$$

Generative Threshold Induction At this stage, we already have visual model Ω and dataset Υ on the training page. The remaining question is that: what would be the appropriate generative threshold for our extraction framework on this particular data source. As discussed above, a better threshold should induce a better output. In the context of this paper, we use F1 as the standard measure to evaluate quality of extracted data records. Let $F_{extract}$ denote the extraction function in Chapter 3.2. $D_i = F_{extract}(\Omega, \theta_i, \mathcal{P})$ is the output dataset when extract page $\mathcal{P}$with generative threshold θ_i w.r.t predefined model Ω. Intuitively, we can cast our generative threshold optimization problem into a search problem of variable $\theta \in [0, 1]$ to maximize quality of output $F1(F_{extract})$. As in this stage, extraction space is the training page $\mathcal{P}_0$, we are going to maximize $F_{extract}(\Omega, \theta_i, \mathcal{P}_0)$. We convert the original search space into a discrete search space $[0, \epsilon, ..., n\epsilon]$ (with $n = \lceil 1/\epsilon \rceil$. By doing this, we always guarantee the error bound ϵ of the induced threshold. For each extracted dataset D_i, F1 measure can be calculated by Equation 4.3. This search problem is straight-forward and will not be presented in detail in our paper.

$$F1(F_{extract}) = \frac{2|D \cap F_{extract}|}{|D| + |F_{extract}|} \tag{4.3}$$

Algorithm 1 Model Induction

1: **Input:** m *clean* clusters (ascending w.r.t vertical coordinate) $E_i^* = \{d_i^k\}$ (k=1..m_i); Training example $(e_{01}, e_{02}, \ldots, e_{0m})$
2: **Output:** output dataset $\Upsilon = \{\Upsilon_i\}$
3: **begin**
4: /* Record candidate pool */ $P_c = \emptyset$
5: /* candidate generation */
6: /* record candidate */ $r \leftarrow (r_1 = null, \ldots, r_m = null)$
7: $expand\_times = 0$
8: /* Candidate-Expansion: */
9: **for** *each cluster* E_i^* *that* $r_i =$ *null* **do**
10: **if** $\exists r_i^* \in E_i^*$ *so that* $(r_i^*.y - e_i.y = r_k.y - e_k.y)$ *and* $(visual\_invariant(r_i^*, r_k) =$ *true*$)$ *with*$\forall$ *k that* $r_k \neq null$ **then**
11: $r_i \leftarrow r_i^*$
12: $expand\_times++$
13: **else**
14: /*avoid regenerate this optional attribute again by changing null value*/
15: $r_i \leftarrow$ *optional*
16: $expand\_times++$
17: **end if**
18: **end for**
19: **if** $expand\_times < m$ **then**
20: *goto* Candidate-Expansion
21: **else**
22: $P_c \leftarrow P_c + r$
23: **end if**
24: /* maximize non-overlapped dataset by top-1 ranking approach*/
25: $\Upsilon = \emptyset$
26: Record Pool $Pool = \emptyset$
27: /* Expansion: */
28: $expand\_pool =$ *set of* $r \in P_c$, r *not overlap any* $r^* \in Pool$
29: **if** $expand\_pool \neq \emptyset$ **then**
30: **for** *each* $r \in expand\_pool$ **do**
31: $Pool \leftarrow Pool + r$
32: *goto* : Expansion(line 27)
33: **end for**
34: **else**
35: /* cannot expand anymore, then check if it is maximal pool*/
36: **if** $|Pool| > |\Upsilon|$ **then**
37: $\Upsilon \leftarrow Pool$
38: **end if**
39: **end if**
40: **end**

Chapter 5

Experiments

In order to assess the effectiveness of our approach, we evaluate it against different approaches over 2Y5D dataset - a huge collection of webpages which guarantees the diversity of representation in both high level (i.e. webpage interface) and low level (i.e. HTML source code). We first study the accuracy of our systems in comparison with other systems. Secondly, we show how robust a high-leveled approach is against low-leveled ones. Lastly, we compare the consistency of our system with that of others.

Dataset The the purpose of evaluation, we use 2Y5D, a dataset collected from Oct-2004 to Aug-2006 on five domains: automobiles, book, job, movies and music. Each domain consists of from 15 to 25 sources and there are several queries applied on each source. For each query, we retrieves up to three result pages returned at one queried time. Figure 5.1 summaries characteristics of this dataset.

5.1 Accuracy and Robustness

To assess accuracy and robustness of our Visual Relational Extraction approach (**Vex**), we compare it against two other approaches: RoadRunner (**RR**) y Crescenzi at al. [1] and **MDR** by Liu et al. [12]. We chose these two systems because tree-alignment approaches the latest and most effective. They can be divided into two categories based on the requirement to extract data: **1**. multiple-page approach which requires at least two pages to align; **2**. single-page approach which executes the alignment on each input page. RoadRunner represents the former and MDR represents the latter. It should be noted that the output of

Domain	#Sources	#Webpages	#Queries
Auto	13	7148	3
Book	25	31693	6
Job	15	9545	4
Movie	23	22822	6
Music	22	23853	6

Figure 5.1: 2Y5D Dataset Characteristics

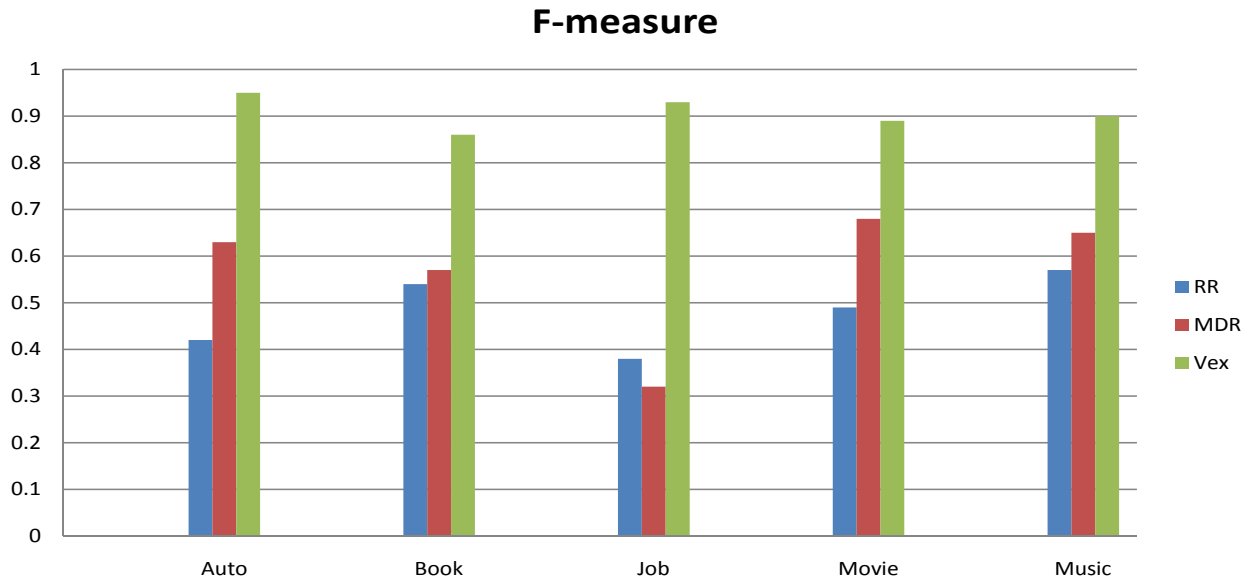

Figure 5.2: F-measure evaluation

MDR system is segmentations of data records only. However, we understand that with the same mechanism and algorithm, they can further extend the current system to recognize all possible properties of each data record. Therefore, in favor of MDR, we assume that all properties of each extracted data record are correct as long as its region is recognized correctly.

Setup As mentioned above, at one specific time in our dataset, there might be either one or several result pages crawled for one combination of source and query. Since RR requires several pages to execute, we automatically retrieved all of the combinations of source, query and time which have at least two pages collected. Each combination in this set, therefore, guarantees that the corresponding data can be applied for all three aforementioned approaches. In our experiment, we took randomly 7 sources on Auto, 12 on Book, 8 on Job, 11 on Movie and 10 on Music for evaluation. For each source, we decided to have only one example in time dimension to avoid duplication in case there is no change in its HTML code over time. For the same reason of avoiding duplication, even one combination of source-query-time contains several result pages, we only run MDR and Vex on one page. Roadrunner, however, is executed on all pages and the final result is the mean of accuracy values achieved from those pages.

Metrics: Extraction accuracy is measured using F-measure (*i.e.*, F1 score) - a harmonic mean between precision and recall. For a set of input pages, let M is a set of correct data records which are manually identified; E is the set of data records extracted by one algorithm, then precision P, recall R and F-measure F are defined as: $P{=}\frac{|M \cap E|}{E}$; $R = \frac{M \cap E}{M}$; $F = \frac{2PR}{P+R}$

Results Figure 5.2 shows the accuracy of all three approaches over five domains. The statistic indicates that Vex outperforms other approaches on all tested domains with very high accuracy ranging from 0.86

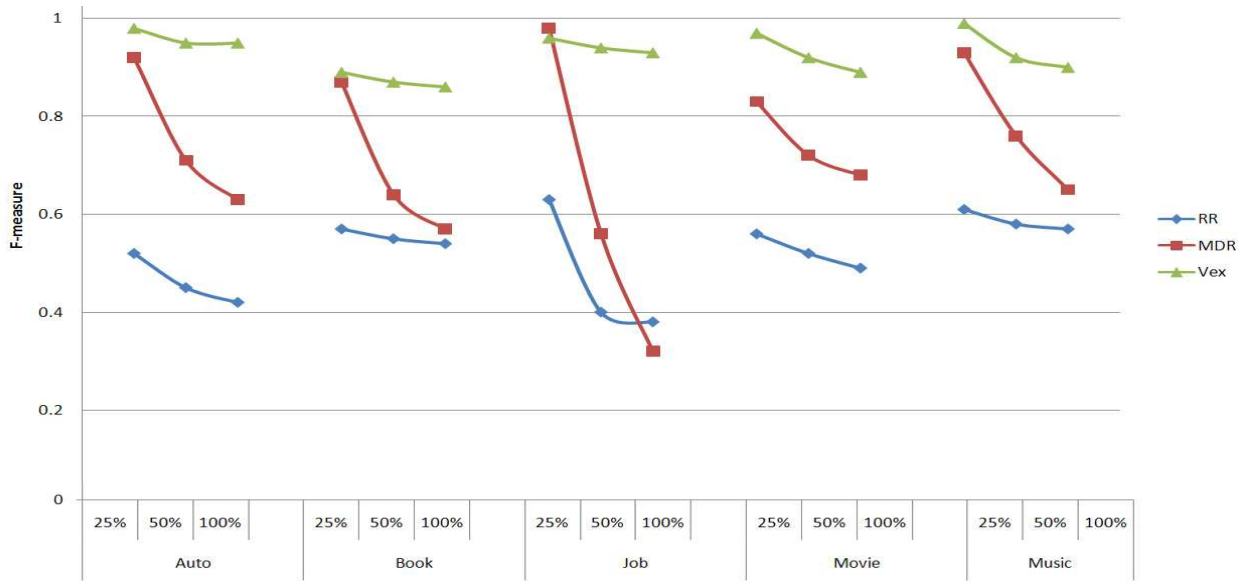

Figure 5.3: Robustness with different webpages structures

to 0.95. Especially on Auto and Job domain, we achieved the highest accuracy with F-measure equals to 0.95 and 0.93, respectively. Examining the structure of the sources on these domains, we notice two possible explanations for this phenomenon. First, most of the sources in these two domains have a very well-structured interface with each attribute is put into one cell of the result table. This type of layout somehow guarantees the consistency of the alignment and thus of the visual relations which seems not even need the tolerance provided by our probabilistic model. The second possible explanation comes the fact that we hardly observe optional attributes insides the sources of those two domains. This may be also the reason why we have the worst result in book domain. In book sources, we frequently see books lacking some of the attributes such as *used Price*, *format* or sometimes even *Authors*. We can see that RR performs relatively bad in our experiment. A closer look at the sources which RR performs the worst, we notice that RR have problem in recognizing recursive structure in the page which in turn, outputs multiple data records as one single tuple. Consequently, the corresponding precision and recall is affected greatly. MDR performs moderate in our experiment. Its result is generally better than RR and worse than Vex.

Figure 5.2 shows a big gap between the average accuracy of our system and that of MDR and RR. This fact holds on all of the tested domains with a large dataset which guarantees the diversity of HTML structures. Does that simply mean that given any input pages, Vex will outperform MDR and RR in term of extraction accuracy? If not, then what is the reason for the gap shown in Fig 5.2? To answer these considerations, we decided to show our collected statistic in another perspective. For each domain, how do RR, MDR and Vex perform in their best input, good input and general input? Figure 5.3 shows F-measure

Domain	Sources S	pages P	#Sampled S	#Sampled P
Auto	13	7148	3	2418
Book	25	31693	5	4923
Job	15	9545	3	2618
Movie	23	22822	5	5036
Music	22	23853	5	5921

Figure 5.4: Consistency Test: Sampled Dataset

statistic of each approach in their top 25% sample pages, top 50% sample pages and all the sample pages, respectively.

Sensitivity to input data - Robustness The result in Figure 5.3 shows a steady drop of MDR's accuracy from top 25% result to average result (i.e. 32% decrement on Auto, 35% on Book, 67% on Job, 18% on Movie and 30% on Music). Not dropped as fast as MDR, RR also has significant decrement from top 25% to 100% result such as 40% on Job and 19% on Auto domain. This indicates that MDR and RR are extremely sensitive to the input data. In top 25% result, MDR's accuracy is generally comparable with Vex's. However, in top 50% result, MDR's accuracy is already significantly lower than Vex's. On contrary with MDR and RR, Vex shows a stable result on all three scenarios, the highest difference of top 25% and 100% accuracy is in Music and Movie domains with a drop of 9% and 8%, respectively. There is virtually no difference in the three scenarios on other domains. This means our approach is very robust on different webpage structures.

5.2 Consistency Over Time

In this part, we do experiment to examine how our generated wrapper can cope with webpage changing/updating over time. Fortunately, our dataset is perfectly matched this type of experiment since we have the data collected from 2004 to 2006 for each source in all five domains. We compare our system with wrapper generated by RoadRunner (**RR**) and a DOM-path-based wrapper model (**DPath**) [7] (MDR also belongs to this type since it uses DOM path to measure similarity - even it does not induce a wrapper explicitly)

Setup In this experiment, we randomly picked 3 sources from Auto and Job domain, 5 sources from each of the other domains. For each source, we collected pages from all the dates available in our repository in a time-ascending order. We first generated wrapper from the first date of this source, run it on consecutive dates until it breaks. Whenever the wrapper breaks, we regenerate it and continue the same process. The

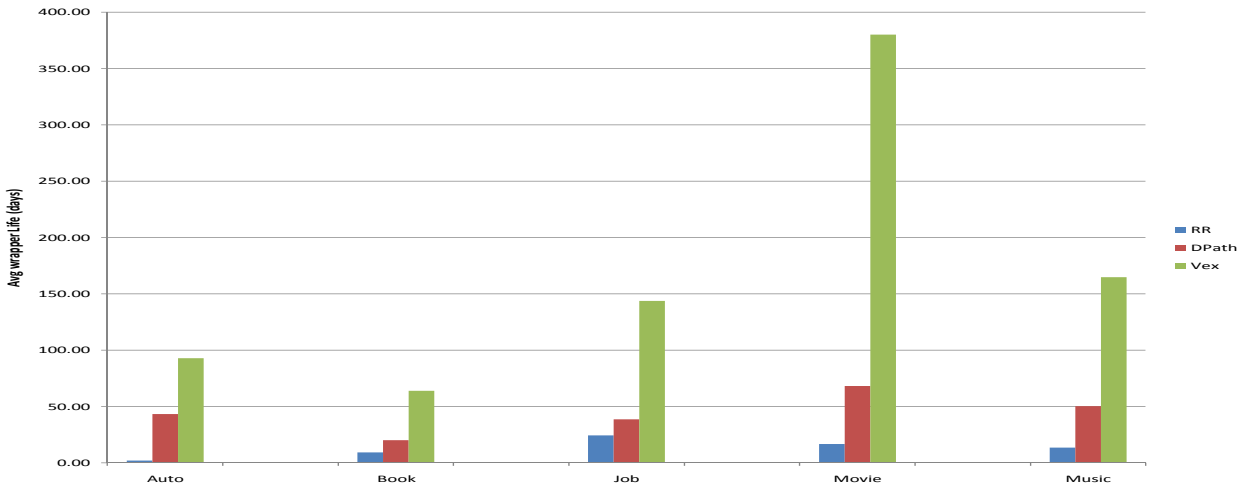

Figure 5.5: Average of induced wrapper's life

average life (days) of these wrappers is the measure represented the consistency of the wrapper against this source. Figure 5.4 shows sampled dataset.

The result in Figure 5.5 indicates that Vex outperforms other approaches in term of consistency of inferred wrappers. It gets the best result in Movie domain. Specifically, in this domain, our experiment shows that 40% of the tested sources (i.e. two out of five) gives the perfect result with Vex. That means the wrapper generated from data collected in the first date in our dataset still works well with that source over almost two years. This also infers that the real lives of those wrappers are actually longer than the values we recorded.

Wrappers generated by RR broke quite frequently. This might result from the fact that RR uses the generated AND-OR tree as its wrapper so that it can align with a new input pages to get the result extracted. However, any small update of the source, which does not even belong to the want-to-be-extracted region, can compromise some AND node and therefore make the whole wrapper broken. DPath improves the model by describing the path to desired elements only. However, whenever there is some addition/removal of tags in those element regions such as format tags, then the DPath rules are violated and thus the corresponding wrapper breaks. Wrapper model of Vex, in the other hand, is built on the highest abstraction level (i.e. visual interface). Therefore, most of the normal changes in HTML source code (even when some new attributes are added into data records) do not affect the visual relation. It explains why Vex greatly outperforms RR and Dpath in wrapper consistency benchmark.

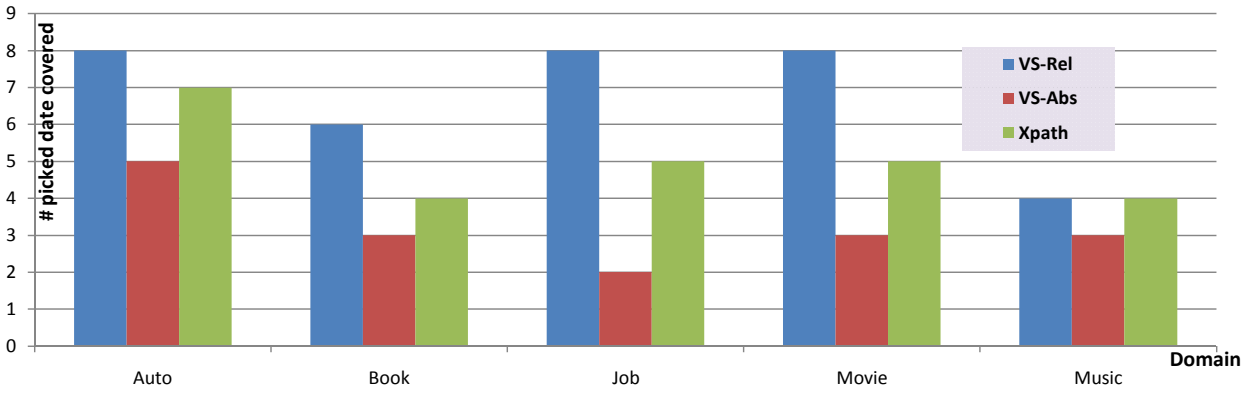

Figure 5.6: Feature Coverage

5.3 Features Effectiveness Evaluation

To illustrate the effectiveness of the features chosen in our model (*i.e.*, visual relation), we try to compare with other types of feature in different perspectives. In this section, we do not intend to discuss about any specific extraction system, only the choice of features used in extraction. Since our feature (denoted by *VS-Rel*) is high level and relative (*i.e.*, it does not describe specification for any attribute but the alignment *between* them), we propose comparison with *low-level* features and *absolute high level* features. The former is represented by XPath from document root (denoted by *Xpath*) since most of the works in this level use DOM-path to express desired attributes. The later is represented by CSS features (denoted by *VS-Abs*). Specifically, VS-Rel uses explicit relations (top, left) and implicit ones (vertical distance between 2 consecutive attributes); Xpath uses path from HTML tag and consider the length as number of features (so we can count similarity of other element by number of held features); VS-Abs uses font name, color and font-weight and font-style for each attributes. In general, the number of features are almost the same on three feature sets (16-18 for 4 attributes).

In wrapper induction topic, regardless of the concrete extraction techniques, good features should be able to cover a good: 1. *Feature coverage* over time which means data record should have the same value of chosen feature over time (thus empower *Consistency* of the system built on top); and 2. *Discriminative power* between good and bad data records of the same source at one specific time (thus empower *Accuracy*). Remember that a system can have better Consistency and/or Accuracy than others even its features have less Feature coverage and/or Discrimination power by applying several heuristics and better techniques in its extraction framework (but it is out of scope of this evaluation).

Feature Coverage on time dimension For each domain, we randomly chose two sources on 2Y5D. For each source, we labeled one data record at the first crawled date and used its features' values as the base

feature values. For the next 30 crawled dates, we picked 4 dates in a relatively equal interval, each date has one data record labeled. Features' values from those records were compared with base feature values. We recorded the number of labeled records which have identical feature values with base feature values. That number is presented in the Y axis of the chart on Figure 5.6. Visual Relation is almost the same on most of the picks. Absolute Visual performs the worst which indicates the most frequent change/update are about Cascade Style Sheet update (*i.e.*, format). Visual relation is perfect on 3 domain and rather bad in Book and Music which we do not expect since the Wrapper life in our system is high. A close look on the bad source on Music (all four picks are failed), we notice the reason comes from the fact that one attribute is missing right after the first date we crawled (*i.e.*, Songs which is a list of songs on the Album) and our evaluation test consider the later records have different features since Song is now *null*. This kind of change, however, does not affect our extraction framework.

Discrimination Power A feature set is considered to have good discriminative power if good data records are highly similar w.r.t feature values while a good and a fake record should have highly different values. For each domain, we randomly picked one source in 2Y5D and one live source on the web since live sources have much more noise than the ones we crawled (missing frames). For live sources, we chose Amazon.com for Book, Hotjobs.yahoo.com for Job, Cars.com for Auto, cdconnection.com for Music and Netflix for Movie. For each source, we labeled one good record then get candidate list for each attribute which is a list of HTML elements with the same tag name with the labeled attribute. We then combined random candidate of each attribute to form a testing tuple. After comparing the features similarity with the labeled record, we categorized tuples into four types of similarity: low (¡40%), medium([40%, 60%)), med-high([60%, 80%)) and high([80$, 100%]). The number of testing tuples on each domains is roughly 25M (millions), 34M, 22M, 8M, 3M on Book, Job, Auto, Music and Movie respectively (total possible tuples are around hundreds millions). Since the test is extensive and number of tuples are huge, we report the distribution on the log chart in Figure 5.7 (the y axis denotes log(N) where N is the number of tuples belong to each category)

In general, VS-Rel shows a very consistent trend on all domains with the majority of tuples has low similarity with labeled record while the number of high-similarity tuples is very limited (maximum is 348 on Movie). Others feature set have unpredicted trend which changes from sources to sources. For Visual-Abs, most of the testing tuples are on med-high or high category except for book domain. A deep look into the format of these sources, we see that on hotjob, netflix or cdconnection, almost all texts and links share the same format (color and font name) except font weight and font style. In Book domain, however, VS-Abs shares the same trend with VS-Rel. It looks like using format features is only applicable in rich-formatted pages which define diverse format for different elements. For Xpath, the expected trend (*i.e.*, indicate good

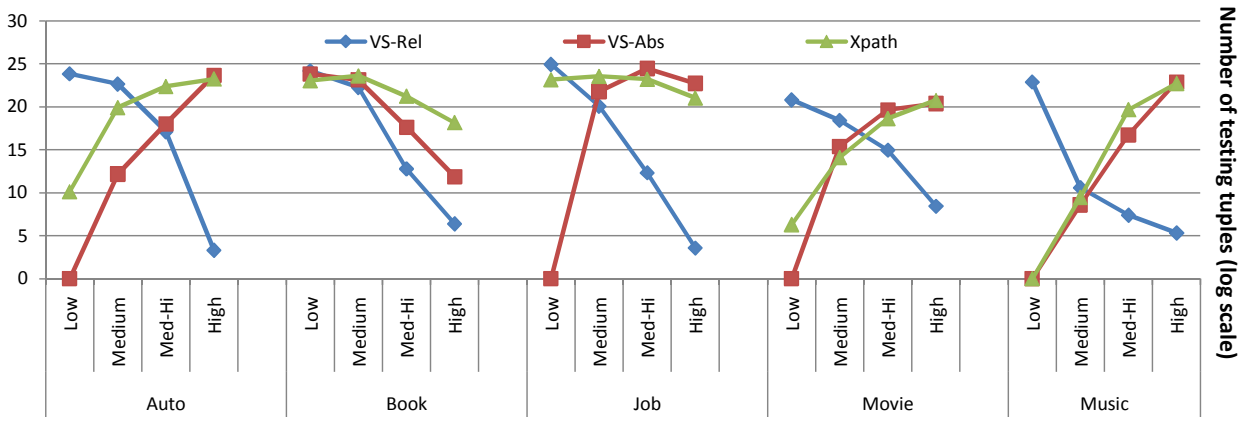

Figure 5.7: Statistics on each Feature Similarity Level ($logN$)

discrimination) is observed in Book and Job but not in the other domains. The structure of tested sources on these domain reveal this fact. For example, on Music (*i.e.*, cdconnection.com empirically), there is almost no noise, the content only contains the result table with different attribute on each records. The majority of testing tuples, therefore, always have high similar structure with labeled one. In Amazon and Hotjob, however, there are a lot of noise for each attributes from different regions of the pages and thus XPath performs better. This evaluation implicitly tells the discriminative power of Visual Relation regardless of the page structure and/or embedded CSS of webpages.

Chapter 6

Conclusion

This paper studies the problem of wrapper generation and proposes the concept of visual-relational data extraction as the foundation for modeling wrappers. Towards large scale integration, we identify the key requirements of wrapper deployment, and observe the limitations of the state of the art— which inherently result from their low-level wrapper modeling. We thus propose the visual-relational modeling and develop the execution and learning mechanisms. Our experiments show significant improvements towards satisfying the accuracy and consistency requirements. For future work, we want to extend the wrapper model to also incorporate the *navigation* of a target data source, beyond data extraction.

References

[1] Mecca G. Crescenzi, V. and P. Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In *Proceedings of VLDB*, 2001.

[2] J. Hammer, H. Garcia-molina, J. Cho, R. Aranha, and A. Crespo. Extracting semistructured information from the web. In *In Proceedings of the Workshop on Management of Semistructured Data*, pages 18–25, 1997.

[3] Minton S. Muslea, I. and C. Knoblock. A hierarchical approach to wrapper induction. In *Proceedings of 3rd Conference on Autonomous Agents (AA-99)*, 1999.

[4] Joachim Hammer, Héctor García-Molina, Svetlozar Nestorov, Ramana Yerneni, Marcus Breunig, and Vasilis Vassalos. Template-based wrappers in the TSIMMIS system. In *Proceedings of ACM SIGMOD*, pages 532–535, 1997.

[5] C.-N. Hsu and M.-T Dung. Generating finite-state transducers for semi-structured data extraction from the web. *Information Systems*, 23:521–538, 1998.

[6] Nicholas Kushmerick, Daniel S. Weld, and Robert Doorenbos. Wrapper induction for information extraction. In *IJCAI*.

[7] Zonghuan Wu Vijay Raghavan Clement Yu Hongkun Zhao, Weiyi Meng. Fully automatic wrapper generation for search engines. In *WWW*, pages 66–75, 2005.

[8] Robert Baumgartner, Sergio Flesca, and Georg Gottlob. Supervised wrapper generation with lixto. In *VLDB*, pages 715–716, 2001.

[9] Weld Daniel S. Doorenbos Robert B., Etzioni Oren. A scalable comparison-shopping agent for the www. *Autonomous Agents*, 1997.

[10] Ling Liu, Calton Pu, and Wei Han. Xwrap: An xml-enable wrapper construction system for web information sources. In *ICDE*, pages 611–621, 2000.

[11] G. O. Arocena and A. O. Mendelzon. Weboql: Restructuring documents, databases, and webs. 1998.

[12] Grossman R. Liu, B. and Y. Zhai. Mining data records from web pages. 2003.

[13] Y. Zhai and B. Liu. Web data extraction based on partial tree alignment. In *Proceedings of WWW*, 2005.

[14] Ji-Rong Wen Deng Cai, Shipeng Yu and Wei-Ying Ma. Extracting content structure for web pages based on visual representation. *Asia-Pacific Web Conference*, 2003.

[15] Kai Simon and Georg Lausen. Viper: augmenting automatic information extraction with visual perceptions. In *CIKM '05: Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 381–388, New York, NY, USA, 2005. ACM.

[16] Zhen Zhang, Bin He, and Kevin Chen chuan Chang. Understanding web query interfaces: Best-effort parsing with hidden syntax. In *sigmod*, pages 107–118, 2004.

[17] Hoa Nguyen, Thanh Nguyen, and Juliana Freire. Learning to extract form labels. *Proc. VLDB Endow.*, 1(1):684–694, 2008.