# Context-Aware Wrapping: Synchronized Data Extraction*

Shui-Lung Chuang schuang2@uiuc.edu
Kevin Chen-Chuan Chang kcchang@uiuc.edu
ChengXiang Zhai czhai@uiuc.edu
Computer Science Department, University of Illinois at Urbana-Champaign

## ABSTRACT

The deep Web presents a pressing need for integrating large numbers of dynamically evolving data sources. To be more automatic yet accurate in building an integration system, we observe two problems: First, across sequential tasks in integration, how can a wrapper (as an extraction task) consider the peer sources to facilitate the subsequent matching task? Second, across parallel sources, how can a wrapper leverage the peer wrappers or domain rules to enhance extraction accuracy? These issues, while seemingly unrelated, both boil down to the lack of "context awareness": Current automatic wrapper induction approaches generate a wrapper for one source at a time, in isolation, and thus inherently lack the awareness of the peer sources or domain knowledge in the context of integration. We propose the concept of context-aware wrappers that are amenable to matching and that can leverage peer wrappers or prior domain knowledge. Such context awareness inspires a synchronization framework to construct wrappers consistently and collaboratively across their mutual context. We draw the insight from turbo codes and develop the turbo syncer to interconnect extraction with matching, which together achieve context awareness in wrapping. Our experiments show that the turbo syncer can, on the one hand, enhance extraction consistency and thus increase matching accuracy (from 17-83% to 78-94% in F-measure) and, on the other hand, incorporate peer wrappers and domain knowledge seamlessly to reduce extraction errors (from 09-60% to 01-11%).

## 1. INTRODUCTION

The emergence of *structured databases* online, the "deep Web," has created a pressing need for large scale information integration. With the scale of the Web, in any domain of interests (*e.g.*, Books, Autos, Real Estate), there are usually a large and increasing number of data sources that provide alternative or complementary information. How can we enable users to access these sources uniformly and effectively? Such *large scale integration*, by bringing together data sources for integrated access, will provide a crucial facility for users to exploit the numerous online databases as a whole, or to choose ones that are particularly useful for them. Many useful applications such as *domain portals*, *comparison shopping*, *meta-search*, and *vertical search engines* will need such techniques to scale up. Clearly, to handle a large number of evolving sources, large scale integration mandates more *automatic* yet *accurate* techniques.

Where is the bottleneck towards scaling up? We believe that *wrapper construction* is a central barrier– We need to build a wrapper for *every* new source, to bring it into the integration system. A *wrapper* performs *template*-based data extraction: It contains a set of template rules for converting the HTML text of a source into structured records. For the mandate of rapid construction, *automatic wrapper generation* techniques (*e.g.*, [2, 5, 22, 23]), which do not require supervised per-source labeling and training, have been studied. They generate a wrapper for *one* given source, by inducing its repetitive HTML structure into template rules. (Such "template"-based extraction sets wrappers apart from *information extraction*, which focuses more on unstructured text without clear templates– Section 6).

However, we observe that, for an integration system to incorporate "many" sources, current techniques fall short in several aspects: An integration system, as Figure 1 shows, must perform some sequential tasks, *i.e.*, querying, extraction, and matching, across a set of some $n$ parallel sources. In both "task" and "source" dimensions, current automatic techniques are not geared for more accurate results:

• *First*, across sequential tasks, a wrapper (as an extraction task) shall consider *peer sources* to facilitate the subsequent matching task, to enhance *matching accuracy*. As we integrate $n$ sources, can we enhance such matching by making wrapper $i$, $\forall i$, to extract data in "consistent" ways as all others? *E.g.*, can we build a wrapper for amazon.com to be "consistent" with bn.com– say, both combining firstname and lastname as one author, or both separating title and edition. Such consistency enables simple 1:1 matching, instead of the challenging complex $m$:$n$ matching.

• *Second*, across parallel sources, a wrapper shall leverage other peer wrappers or domain rules to enhance *extraction accuracy*. For domain-based integration, as we focus on the same domain of sources (*e.g.*, Books), can we supply some domain "knowledge" as a *reference model* to guide wrapper $k$– *e.g.*, regular-pattern rules specifying that isbn *must* have 10 digits, or a statistical model describing title? Fur-

*This material is based upon the work partially supported by NSF Grants IIS-0133199, IIS-0313260, the 2004 and 2005 IBM Faculty Awards. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the funding agencies.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, to post on servers or to redistribute to lists, requires a fee and/or special permission from the publisher, ACM.
*VLDB '07,* September 23-28, 2007, Vienna, Austria.
Copyright 2007 VLDB Endowment, ACM 978-1-59593-649-3/07/09.

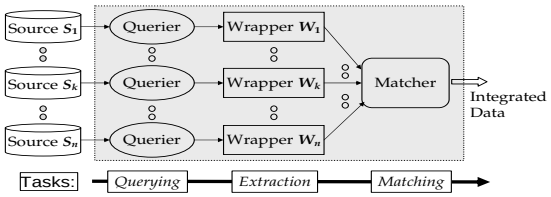



**Figure 1: An integration system with $n$ sources.**

ther, as we *incrementally* build wrappers one after another, after having $k$ wrappers, can we build wrapper $k$+1 (say, bn.com) by leveraging wrappers $1, \cdots, k$ (*e.g.*, amazon.com, borders.com) to correct errors? The ability to leverage prior knowledge or wrappers enables us to reuse preceding efforts towards incrementally building up large scale systems.

**Our Proposal: Context-Aware Wrapping.** These issues, while seemingly unrelated, boil down to the same lack of "context awareness": Current wrapping approaches generate one wrapper at a time, *in isolation*, fundamentally lacking the awareness of peer sources or domain knowledge– which we refer to as the *context* of integration. Without context awareness, current wrappers can neither build upon domain knowledge nor align with peer sources, resulting in inaccuracies in both extraction and matching. How can we exploit the rich context, in terms of the clear domain focus and abundant peer sources, in large scale integration?

To visualize the opportunities, as Figure 2 shows, suppose we are integrating a particular domain (*e.g.*, Books), and we want to build wrappers for some new *peer sources* (*e.g.*, bn.com and buy.com). At this point, we may already have wrappers for others, which we call *reference sources* (*e.g.*, amazon.com, borders.com), and we may have a reference model (*e.g.*, isbn rules). Our objectives are to 1) leverage reference sources and the reference model, and 2) consider peer sources to form consistent extractions.

We believe the key to unifying both goals is to broadly abstract "context" as a set of "generalized sources"– not only the real peer sources (which we are to wrap consistently) but also the reference sources and model (which we are to leverage). Note that a source is, for our purpose, capable of producing data records in the domain of interest: *e.g.*, bn.com will generate, say, [title, price, ...]. To unify, we conceptually view a reference source or model as also a "generator," or a *generative model*, of some fields (*e.g.*, isbn). We can then exploit the rich context by seeking the consistency between all the "sources," which will thus achieve both objectives.

With this view, this paper proposes a new problem: How to realize *context-aware wrapping*? We identify context awareness as a necessary concept for wrapper generation among multiple sources, in order to enhance both extraction and matching accuracy, as well as to reuse wrapping efforts. While Section 3 will formally define the problem, we sketch it as follows, referring to Figure 2:

• The *Context-Aware Wrapping* Problem: Given some *new sources* $N = \{S_1, \cdots, S_n\}$, possibly some *reference sources* $R = \{S_a, \cdots, S_b\}$, and possibly a *reference model* $M_\gamma$, build one wrapper $W_k$ for each new source $S_k$, so that the extractions from all sources in the context (*i.e.*, $N \cup R \cup \{M_\gamma\}$) are the most *consistent*. ■

**Approach: Synchronized Extraction.** Our goal is thus to maximize context consistency in data extraction. As Figure 2 shows, in the wrapper generation process, starting with sample data pages (*e.g.*, query result pages to wrap) at step 1, if we can obtain consistent extractions, as step 3 shows, we can then use these data records as labeled examples to

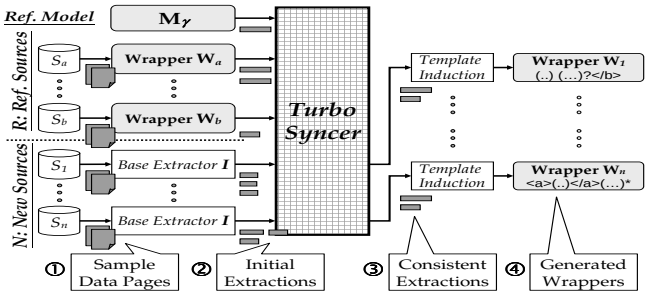



**Figure 2: Context-aware wrapper generation.**

perform template induction (*e.g.*, [10]) to generate the template rules of the final wrappers. The key challenge is: How to form consistent extractions? (Step 2 to 3.)

While traditional wrapper generation works *in isolation*, our approach is thus "synchronized extraction," where each wrapper should segment data fields consistently, or *in sync*, with the context. Towards synchronization, can we also leverage current "per-source" wrappers as the basis?

Our solution is thus a middle layer, as a *syncer* (Fig. 2), to synchronize the initial extractions from some *base extractors* $I$– which can be any current per-source wrapper– into a consistent state, before feeding into template induction.

**Insight: Turbo Decoding.** In search of the insight, we realize that such synchronized extraction can be regarded as a multi-code *communication* process: As a source renders its data, or *encode* its message, our objective is to extract the data, or *decode* the message. With this analogy, we draw our insight from *turbo codes* [3] in information theory. The intuition is simple: In transmitting, the *turbo encoder* will dispatch the same message through multiple error-independent "codes," and thus, in receiving, the *turbo decoder* will recover the message by synchronizing between these codes. We thus develop synchronized extraction, *Turbo Syncer*, as synchronized decoding, which then imply an *EM*-style (Expectation-Maximization [8, 6]) iterative optimization framework. Thus, by viewing sources as encoding the same message, we enforce their extractions, as in turbo decoding, to synchronize to the common message.

We have implemented the Turbo Syncer and extensively evaluated it over 30 sources in 3 domains, using 4 base extractors [2, 5, 22] and 3 matchers [18, 21], across various desired "context-aware" settings: Wrapping multiple sources concurrently, and leveraging reference sources and user-crafted reference models. It indeed reduces the extraction errors (from 9–60% to 1–11%) and raises the matching performance (from 17–83% to 76–94%). Further, it is robust, raising the accuracy to a rather constant level, regardless of the base extractors.

In summary, this paper makes these contributions:

- *Concept and Problem*: We propose the novel concept of **context-aware wrapping** and thus the new problem of its realization, to support large-scale integration systems.
- *Framework*: We propose synchronized extraction and develop the **Turbo Syncer**, inspired by turbo codes.
- *Evaluation*: We empirically validate that synchronized extraction is effective in various **context-aware settings**.

In the rest of the paper, we start with motivating synchronized extraction (Section 2), followed by the turbo framework (Section 3), the detailed algorithm (Section 4), and the empirical evaluation (Section 5). The related work (Section 6) and conclusion (Section 7) are then drawn out.

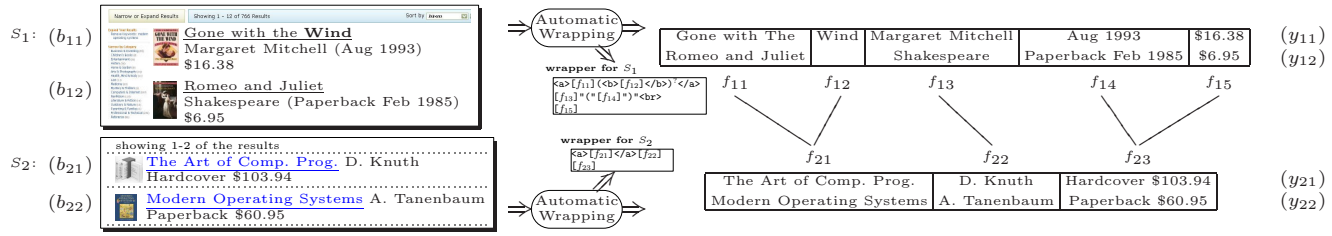



**Figure 3: A simple example of two book sources, each with two records.**

# 2. MOTIVATION

This section motivates our thesis: the concept and realization of context-aware wrapping. To start off, we reiterate the two objectives: 1) Consider peer sources to enable consistent extractions in order to enhance matching accuracy. 2) Leverage multiple "sources" to enable cross-correction in order to enhance extraction accuracy. (Recall that Section 1 unifies both reference sources and models as such sources.) As our motivation, we ask– *First*, does extraction inconsistency indeed compromise the subsequent matching? *Second*, how is cross-correction possible to achieve consistency and accuracy? This section answers both questions.

## 2.1 Cause: Inconsistency

We address the first question on how extraction impacts matching. We note that a template-based wrapper relies on the structure regularity (or skeleton) to extract the potential data fields of the source, using template rules (*e.g.*, delimiter strings or regular expressions) induced from HTML tag analysis. While wrappers themselves capture the page *syntax* regularities which are source-dependent, the subsequent matching is mainly determined based on their extracted *content* from the input pages. Hence, it is affected if the extraction has inconsistencies. To reveal the inconsistencies and their impacts on matching, we conduct some preliminary observations. We begin with an example:

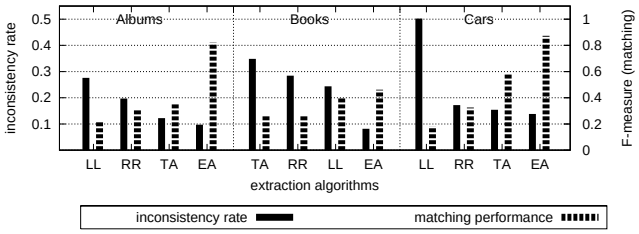
**Example 1 (Book Sources):** Let's suppose a simple scenario of only 2 sources, each with 2 records (as Fig. 3). Also, we suppose there is an automatic wrapper generator which accepts input pages from a source, induces the wrapper, and extracts the enclosed data records.

After wrapping, the two raw records $b_{11|12}$ in source $S_1$ are extracted as structured records $y_{11|12}$ with 5 fields $f_{11}$–$f_{15}$. Similarly, $b_{21|22}$ in $S_2$ become $y_{21|22}$ with 3 fields $f_{21}$–$f_{23}$.

The correspondence (*i.e.*, *best matching*) between the data fields of $S_1$ and $S_2$ (performed by human to match the fields) is then shown as the links between $f$'s in Fig. 3. ■

Notice that, to qualify the extracted fields (so human can match the fields that have overlapping semantics), there exist basic concepts for the given domain. For example, a book consists of title, authors, pub-date, format, etc. A concept may have sub-concepts– *e.g.*, an author has firstname and lastname, and pub-date contains month and year. We regard all these concepts (with different granularities) in mind as *semantic units*. Ideally, the fields with overlapping semantic units should be matched– *e.g.*, in Example 1, $f_{14}$ and $f_{23}$ are matched because they both contain format.

A reasonable extractor seeks to split along the boundaries of semantic units. However, it is often imperfect. A *fragment* occurs when it splits a semantic unit into pieces that are not sub-concepts (*e.g.*, $b_{11}$'s title is broken into "gone with the" and "wind" in $y_{11}$). A *compound* occurs when it does not split at the boundary of two different semantic units (*e.g.*, $f_{14}$:format|pub-date in $S_1$).




LL: LineSpliter ; RR: RoadRunner ; EA: ExAlg ; TA: TreeAlign

**Figure 4: Inconsistency vs. matching.**

While a fragment is always counted as an *inconsistency* (against the semantic unit), a compound may cause an *inconsistency* if the semantic units in it do not co-occur adjacently in other sources. For example, the extracted field $f_{14}$:format|pub-date in $S_1$ is considered inconsistent because format in $S_2$ is not adjacent to any pub-date information.

To show how inconsistencies affect matching in real situations, we conducted a preliminary experiment: We applied 4 automatic Web data extractors– LineSplitter (Sec. 5), RoadRunner[1][5], ExAlg[2][2], and Tree-Align[3][22]– on 3 domains, each with 10 sources (see Sec. 5 for the detailed setup). We checked the numbers of inconsistencies for every extracted record. In addition, we manually compiled, for each domain, an integrated table with no inconsistent fields, and then used it to evaluate the result of doing best-matching on the extracted data. Figure 4 depicts, for the 3 domains with 4 extractors, the average inconsistency rate for a record, and the final matching performance (via F-measure).

In Fig. 4, we observe that the existing extractors produce many inconsistencies (22% in average, where 5% are fragments and the rest 17% are compounds). They have a negative correlation with the matching performance– it usually has a higher performance when there are less inconsistencies.

Hence, when we consider integration, the extraction should not be considered in isolation. Because the inconsistencies (which significantly impact the matching performance) can only be determined in the context of all sources in integration, extraction of them should be considered together.

Intuitively, no matter how the data are extracted, a perfect subsequent matching can align related fields together to form a correct integrated table– *e.g.*, the fragment errors $f_{11}$ and $f_{12}$ are merged (thus recovered) when matching to $f_{22}$ in Example 1. However, to do this requires *complex m:n* matching, which has a much larger search space than simple 1:1 matching and so far remains difficult in practice. Furthermore, for the inconsistent compounds, since matching

[1]RoadRunner is a representative automatic wrapping algorithm that exploits page layout structure and is publicly available online at `http://www.dia.uniroma3.it/db/roadRunner/`.

[2]ExAlg builds a similar concept as RoadRunner, with more enhanced representation. The program we used to perform the experiment is obtained from its authors.

[3]TreeAlign is our re-implementation of the idea using tree alignment proposed in DEPTA [22].

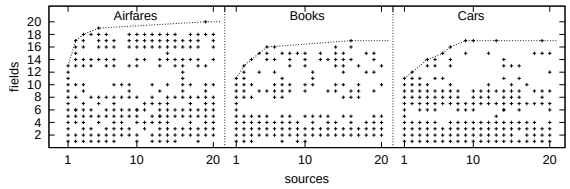



**Figure 5: Field growth on 3 domains.**

only performs merging, the best it can achieve is to group those fields with overlapping semantics, thus forming larger compounds. This may produce a coarse integrated table– *e.g.*, some output field is mixed by many units. Hence, for matching to produce a fine-grained integrated table, extraction of consistent fields is essential.

As implications, since inconsistencies bring in difficulties to align fields and produce a fine-grained integrated table, eliminating them should boost the overall integration.

## 2.2 Opportunity: Cross Correcting

We now move on to the second question on how cross-correction can enhance consistency (and accuracy) of extraction. While the existing approaches build wrappers isolatedly using the syntax regularities within each source, in order for synchronization to align common semantic units between sources to make them consistent, we attempt to gain insight by examining what they share in common and how they differ in their extraction.

First, we survey what they share in common:
*Concerted Convention*: We informally surveyed 3 domains (airfares, books, and cars), each with 20 sources. We manually identified the fields that occurred consistently among all sources. As a result, the average number of different fields in a domain is 18, and in a source is 9.48 (9.48/18=53%). There is 71% overlapping (∼6.72 same fields) between any two sources. And, a field appears in 52% (10.48 out of 20) of sources, in average. These observations indicate: From the perspective of source, each source represents a medium-size *subset* of the domain, and has *high overlap* with other sources. From the perspective of field, same fields occur in many sources, *i.e.*, *high field occurrence*.

Figure 5 further plots the co-occurrence of sources and fields– a "+" at $(x, y)$ means that field $y$ occurred in source $x$. The curves show a *rapid field convergence* (*e.g.*, fields found after the first 5 sources are a:19, b:16, and c:14). ■

Then, we look at how they differ in their extractions:
*Complementary Extraction*: We examined the extraction results in our preliminary experiment (see Section 2.1). Figures 6a–c plot the results of three extractors on book sources– a ∘ at $(x, y)$ means that field $x$ in source $y$ is successfully extracted, and a × means $x$ is in a fragment or inconsistency.

Clearly, none can perform uniformly well for all sources. Even we imagine there exists a perfect ensemble that accepts the 3 approaches (*e.g.*, merging Figure 6a–c as 6d) and takes the best decision for each field (*e.g.*, in 6d, a field with mark ⊗ could be regarded as correctly extracted)– still there are many errors (those marked with × in Figure 6d).

Although neither sources nor fields can be perfectly processed by current extractors, we observed that most sources have *some* fields successfully extracted, and most fields can have a successful extraction in *some* sources. ■

For our objective of synchronization, these observations clearly imply: First, the sources share common fields that need to be synchronized. Second, the complementation between the extractions of sources makes synchronization possible by cross-referring multiple sources. Thus, can we cross-correct the inconsistencies?

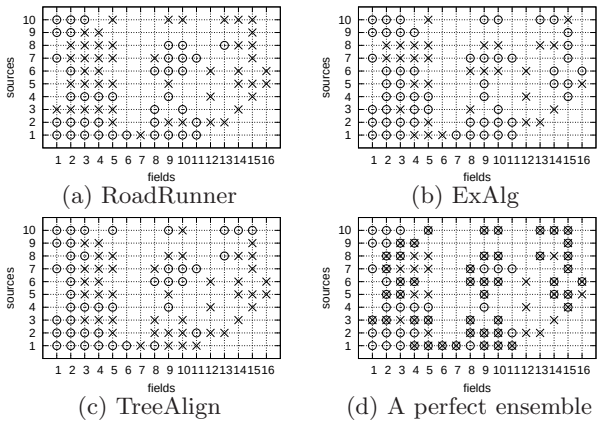



**Figure 6: Existing extractors on 10 book sources.**

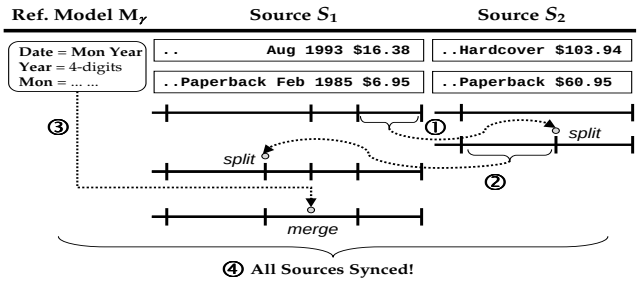



**Figure 7: Synchronization: Cross correction.**

**Example 2 (Cross Correction):** Referring to Fig. 7, suppose we want to synchronize records $y_{12}$–$y_{22}$ (of Fig. 3), which we show only partially. To make the situation interesting, we also assume a reference model $\mathcal{M}_\gamma$ that specifies how date looks like. Fig. 7 shows the field "boundary" positions of the records in each source, and how these boundaries are synchronized in three iterations. Step 1 first propagates the *price* from $S_1$ to $S_2$, resulting in its split of the combined field format+price. In Step 2, as now format is correct in $S_2$, it then feeds to $S_1$ to similarly split its format+month. In Step 3, the year and month are *merged* by the synchronization triggered from the reference model. At this point, all the "sources" ($\mathcal{M}_\gamma$, $S_1$, and $S_2$) are in agreement with their common fields, and thus the context is "synced." ■

As this example shows, we believe the insight of cross correction is promising, by iteratively removing inconsistency errors. As our survey indicates that there are *high field occurrence* and *overlap* among sources– *i.e.*, the chance to have a correct cross reference is high. Overall, the implications show the possibility of synchronized extraction– *e.g.*, an error in a source might be recovered by referring to other sources. As the observations give us a promising insight, we ask: What would be a principled framework to realize it?

# 3. TOWARDS TURBO SYNCER

In this section, we develop our approach to cross-correcting the wrapping inconsistencies between different sources in a turbo-decoding paradigm. We start with the formal problem description (Section 3.1), draw the inspiration from turbo decoding (Section 3.2), resolve the challenges (Section 3.3), and develop our Turbo-Syncer framework (Section 3.4).

## 3.1 Problem Description

Upon the framework in Fig. 2, we seek a consistent wrapping between all the sources through synchronization. Each new source (source to construct wrapper) is first processed by a base extractor, which any existing automatic approach can serve as (*e.g.*, [2, 5, 22]). Basically, the base extractor applies sophisticated syntax analyses on the sample pages of a source (obtained by submitting a few sample queries) to identify the data records/fields, based on which wrappers are generated. Our syncer takes this output (initial structured records produced using per-source syntax regularities), refines it (cross-corrects the boundaries by synchronizing the contents with other peer sources), and generates the newly-structured records for re-training the wrappers.

We now formalize this. Suppose the new sources to wrap are $N = \{S_1, S_2, \ldots\}$. The system may already have reference sources $R = \{S_a, S_b, \ldots\}$, which have their own wrappers (as $\{\mathcal{W}_a, \mathcal{W}_b, \ldots\}$) to perform extraction. Also, there may exist a reference model $\mathcal{M}_\gamma$, storing any user-crafted field knowledge (rules and templates). Notice that for reference sources $R$ and model $\mathcal{M}_\gamma$, while their fields are correctly identified, they may not be complete– *i.e.*, not describing all fields in the domain (referring to Section 2.2, a source contains a subset of fields in the domain). Generally, as Section 1 mentioned, we refer all the input, $N$, $R$, $\mathcal{M}_\gamma$, as sources, since they all generate contents of some fields.

For each source $S_s$, the appropriate extractor (if $S_s$ is a new source, the base extractor; if it is a reference source, its own wrapper) is applied on the sample pages to obtain a set of structured records.

The fields of a record can be specified by their boundary positions. For a record $x_i$ with $\ell_{x_i}$ tokens: $x_i = (x_{i,1}, \ldots, x_{i,\ell_{x_i}})$, if it has $k$ fields, the boundary points are specified by a *segmentation* $y_i = (b_0, \ldots, b_k)$ where $b_0 = 1$ (the start point), $b_{i-1} < b_i$, and $b_k = \ell_{x_i} + 1$ (the end point). The $j$-th field is the subsequence $x_i[b_{j-1}{:}b_j - 1]$.

**Example 3 (Segmentation):** The record $b_{11}$ in Fig. 3 has 9 tokens $x_{11} = (\text{gone}, \text{with}, \ldots, \$16.38)$. One possible segmentation is $y_{11} = (1, 4, 5, 7, 9, 10)$, which specifies 5 fields, with the first field $x_{11}[1{:}(4\text{-}1)]$="gone with the." ∎

Since synchronization only changes the boundaries of the extracted records, we separate the contents and boundaries: A record is writen as $(x_i, y_i)$ and the record set of each source $S_s$ is $(X_s, Y_s)$ where $X_s = (x_1, x_2, \ldots)$ and $Y_s = (y_1, y_2, \ldots)$.

Given a content sequence $x_i$, the total possible segmentations (at least $k = 1$ segment and at most $k = \ell_{x_i}$ segments) can be described as:

$$\mathcal{Y}_{x_i} = \{(b_0, \ldots, b_k) : k \geq 1 \wedge 1 = b_0 < \cdots < b_k = \ell_{x_i} + 1\}.$$

For an entire record set $X_s$, the segmentation space is:

$$\mathcal{Y}_s = \{(y_1, \ldots, y_{|X_s|}) : y_i \in \mathcal{Y}_{x_i} \text{ for each } x_i \in X_s\}. \quad (1)$$

While, for each new source $S_s$, the base extractor produces the initial structured records $(X_s, Y_s)$, our syncer seeks a re-segmentation $Y_s'$ so that the newly-structured records, *i.e.*, $(X_s, Y_s')$, are more consistent with other sources. Our problem is thus described as:

$$\textbf{input: } N = \{(X_1, Y_1), \ldots\} \quad R = \{(X_a, Y_a), \ldots\} \quad \mathcal{M}_\gamma$$

$$\textbf{output: } \left\{Y_1', Y_2', \ldots\right\}$$

$$\textbf{s.t. } \max_{\substack{Y_s' \in \mathcal{Y}_s \\ s=1,2,\ldots}} G\left((X_1, Y_1', ), (X_2, Y_2'), \ldots, R, \mathcal{M}_\gamma\right), \quad (2)$$

where $G$ is an objective function measuring the consistency between sources, which will be realized later (Section 3.4).

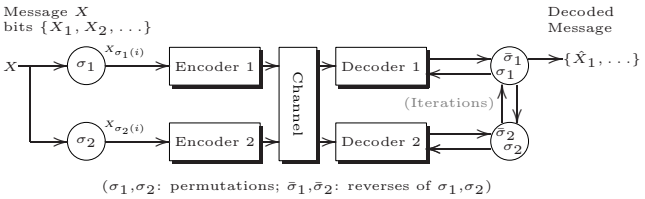



**Figure 8: The framework of turbo codes.**

## 3.2 Paradigm: Turbo Decoding

As mentioned in Section 1, wrapping can be regarded as a *communication* process: It aims to decode the code (HTML pages) produced by the encoder (data source). The information to be decoded are the boundaries of the data fields in the pages. To illustrate, we can view a segmentation as a bit string indicating whether to split at a boundary point or not, thus bringing in analogy to the conventional decoding.

**Example 4 (Bit Rep.):** For $y_{11}$ in Example 3, we can view it as a bit string (1,0,0,1,1,0,1,0,1,1), where a boundary point is split if its bit is 1 and not if it is 0. ∎

As we understand that wrapping is decoding in essence, in order to cross-correct the decoding inconsistencies (as motivated in Section 2.2), how can decoding leverage its "context," instead of performing isolatedly?

With the data from multiple sources (codes from multiple encoders), viewing them as concurrent decoding, we naturally learn from multi-code decoding in information theory and draw the insight from the success of turbo codes [3]. Fig. 8 depicts the framework for turbo codes: The transmitted message bits are passed to two (or more) encoders to produce two codes. Upon receiving, turbo decoders reinforce their decision of each bit by synchronizing their decoding of the received codes. Since the inputs to encoders have been scrambled independently, the chance that both decoders simultaneously produce errors for the same bit is extremely low, and thus the decoding achieves high accuracy.

In our problem, it is apparent that different sources share concerted conventions (analogous to the common message) and produce complementary extractions (analogous to independent errors) (Section 2.2). Turbo decoding hence gives us an encouraging direction.

## 3.3 Challenges: Message & Reinforcing

While turbo decoding sheds a new paradigm for our problem, to apply it, there are two requirements: First, in turbo codes, there is, *by design*, a single identical message embedded in different codes. Second, the correspondence between the message units (bits) in different codes are prescribed, based on which decoders can reinforce each other. These requirements amount to two challenges in our problem:

- *Message*: The data records delivered from different sources are far from identical. As the first challenge: What is the message delivered?
- *Reinforcing mechanism*: The correspondence between the fields (message units) in different sources is not clear. As the second challenge: Without prescribed field correspondence, how can decoders reinforce each other?

### 3.3.1 Message: Hidden Domain Model

To reveal what is the message delivered, we ask: What can be the information shared among same fields across different sources? Consider books $b_{11}$–$b_{22}$ in Fig. 3 and $b_{13}$:

($b_{13}$): [ Come with the Sand John Smith October 2010 $12.34 ]

It is not difficult to tell which subsequences look more like title and authors, even when $b_{13}$ is not a real book.

Inspired from the above examples, what constitutes our recognition is the common information they share– *e.g.*, title is a sequence of words, and authors are personal names. We thus regard such common sharing as the message.

To characterize this sharing (the message) so that similar data can be identified, there have been many heuristics designed, *e.g.*, using character or word distributions, average string length, data types, etc. Since these heuristics are various and ad-hoc in nature, to make our framework flexible, we use a statistical model to unify them. Thus, we hypothesize a hidden schema model that probabilistically generates data records, across all sources in a domain.

**Definition 1 (Statistical Schema Model):** A schema model $\mathcal{M}$ is a finite set of fields $\{f_1, f_2, \ldots\}$, where each $f_i$ is a statistical model specifying how to generate an instance for that field. ∎

**Example 5 (Book Model):** A handcrafted book schema model may look like $\mathcal{M} = \{f_{ti}, f_{au}, f_{fm}, \ldots\}$, where $f_{ti}$ generates title instances (*e.g.*, 2-20 randomly picked-up words), $f_{au}$ generates author(s) instances (*e.g.*, a few randomly picked-up non-dictionary words), $f_{fm}$ generates format instances (*e.g.*, 1-2 words selected from a small vocabulary set {paperback, hardcover, ...}), and so on. ∎

Notice that we are not proposing any new similarity analysis for comparing fields. Instead, we plan to leverage existing approaches in a principled statistical model.

While our hypothesis is conceptual, it is indeed reasonable. As our observations (Section 2.2) shows, the fields in a domain converge quickly to a limited set (the existence of $\mathcal{M}$) and each source is a subset of domain fields.

Thus, with this hypothesis, we can give a simple but fairly principled abstraction of encoding records as pages. For a source $s$: First, a subset of fields in the domain model $\mathcal{M}$ is selected as a source model $\mathcal{M}_s$. Second, data records $R_s$ are generated based on $\mathcal{M}_s$. Finally, records are fed into *Source* $s$ to be rendered as a Web page. This procedure is sketched as the encoding part (left part) of Fig. 9.

Our message is thus the "schema model." The underlying model is not itself the information transmitted, but implicitly carried through by its substantiated records.

### 3.3.2 Reinforcing Mechanism: Matching Measure

To reveal what can be the reinforcing mechanism, we ask: How to determine the correspondence of fields (*i.e.*, message units) between sources? This question brings us to the subsequent matching (Fig. 1). If the extracted fields between sources cannot match well (using some matching measure)– *i.e.*, not consistent, then we have to refine them, which may take several iterations in between sources (like Example 2).

This inspiration answers the challenge: For decoders to reinforce each other, they must incorporate some matching measure and exchange the degree of matching between their outputs, so that their decisions can be further refined.

Note that the notion of matching here is not exactly the standard schema matching problem, which aims at determining a hard decision of *how* fields are aligned. Instead, it only requires producing the scores of *how well* fields between sources are matched. With these scores, the framework will iteratively "re-extract" to form new field boundaries for better extraction, *i.e.*, achieving better matching scores. (Note that in standard schema matching, fields are given and will not be dynamically changed.) In this work, we will develop a matching measure upon our assumption of an underlying model (Sec. 4, Implementation 3). However, our abstraction allows us to apply other matching techniques (*e.g.*, [18]) that are instance-based and can output a similarity score as such matching (*i.e.*, consistency) measure.

This insight naturally motivates an optimization process of finding the extractions among sources to maximize a given matching measure (that is, $G$ in Eq.(2)). The detailed algorithm will be presented later in Section 4.

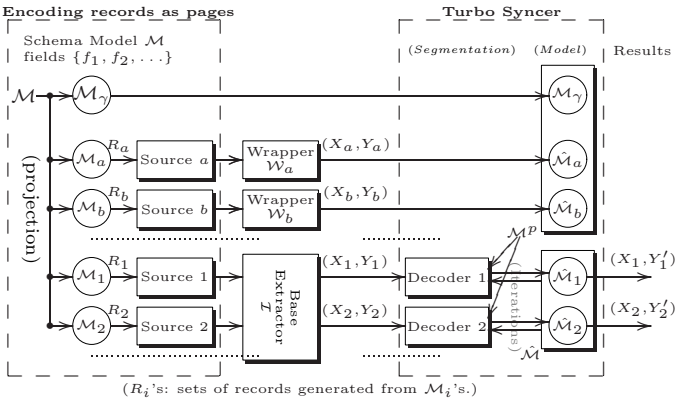



**Figure 9: Our framework of Turbo Syncer.**

## 3.4 Framework: Turbo Syncer

Realizing the message as schema model and the reinforcing mechanism as matching, we now abstract out the problem of synchronization in our context-aware wrapper construction as a turbo-decoding setting, and thus concretize our framework, named *Turbo Syncer*.

Fig. 9 shows the framework. The data pages from sources are viewed as generated by an encoding process from an underlying model (Section 3.3.1). With this conceptual view, we design Turbo Syncer to collaborate: First, the pages from each new source (which has no wrapper yet) are processed by the base extractor. For, if there is any, each reference source (which already has wrapper), its pages are processed by its own wrapper instead. Second, each new source $S_s$ is associated with a *Decoder* $s$, which accepts the initial extraction (from the base extractor) and decodes the source model $\hat{\mathcal{M}}_s$ (as the initial decoding output for $S_s$, using the model training procedure stated in Section 4). The same procedure is applied to construct the models $\{\hat{\mathcal{M}}_a, \hat{\mathcal{M}}_b, \ldots\}$ for reference sources. Since the reference sources are correctly extracted (by their own wrappers), their models are viewed as correct– do not need to be refined. Third, each *Decoder* $s$ takes these models (including the correct models $\mathcal{M}^p = \hat{\mathcal{M}}_a \cup \cdots \cup \mathcal{M}_\gamma$ and the uncertain models $\hat{\mathcal{M}} = \{\hat{\mathcal{M}}_1, \ldots\}$) to reinforce the extraction (by measuring its matching to $\mathcal{M}^p$ and $\hat{\mathcal{M}}$) and update the corresponding source model. The whole process may run for several iterations, to achieve a stabler result.

With the turbo-decoding abstraction, we thus develop an EM-style optimization (the core decoding technique of turbo codes), building upon two principles: (1) From models ($\hat{\mathcal{M}}$ and $\mathcal{M}^p$) to a *Decoder* $s$: All the models in the context ($\hat{\mathcal{M}}$ and $\mathcal{M}^p$) will drive $s$ to refine its segmentation (*i.e.*, adjust its decoding). (2) From *Decoder* $s$ to models: After refining its segmentation, *Decoder* $s$ will output a new model $\hat{\mathcal{M}}_s$ as the decoding result. We then formulate the dual functions of the *Decoder* in a probabilistic framework:

- *Principle 1: (Model to Segmentation)*: Given models $M$, for a source $S_s$, *Decoder* $s$ refines the segmentation as finding a $Y_s^*$ with the maximum likelihood $p(X_s, Y_s^*|M)$:

$$Y_s^* = \arg\max_{Y_s \in \mathcal{Y}_s} p(X_s, Y_s|M) \quad (3)$$

  Conceptually, $p(X_s, Y_s|M)$ encodes how likely $(X_s, Y_s)$ is generated by $M$, thus representing the consistency measure of $(X_s, Y_S)$ and $M$ (implemented in Section 4.1).

- *Principle 2: (Segmentation to Model)*: Given a segmentation $Y_s$, *Decoder* $s$ decodes the model that best generates $(X_s, Y_s)$ as finding a $\hat{\mathcal{M}}_s^*$ with the maximum likelihood

$$\hat{\mathcal{M}}_s^* = \arg\max_{m} p(m|X_s, Y_s) \quad (4)$$

  Conceptually, $p(m|X_s, Y_s)$ denotes how likely a model $m$ is derived from $(X_s, Y_s)$. It is not explicitly computed but implicitly embedded in the model training algorithm.

Upon this probabilistic framework, we hence can concretize the objective function $G$ in Eq.(2) as

$$G\left((X_1, Y_1', ), \ldots, R, \mathcal{M}_\gamma\right) = \sum_{s=1,2,\ldots} p(X_s, Y_s'|\hat{\mathcal{M}}, \mathcal{M}^p) \quad (5)$$

where each source model $\hat{\mathcal{M}}_s$ in $\hat{\mathcal{M}}$ is derived from records $(X_s, Y_s')$ (via Eq.(4)) and $\mathcal{M}^p$ combines models from the reference sources $R$ and model $\mathcal{M}_\gamma$. The details of matching measure $p(X_s, Y_s'|M)$ will be described in Section 4.

Overall, with the dual behaviors defining a *Decoder*, the Turbo Syncer is simply a set of such *Decoders* $1, \ldots, n$, one for each new source, to synchronize and decode iteratively till stabilization (maximizing function $G$).

# 4. ALGORITHM

We now explain Turbo Syncer, as it is a set of decoders, by developing the algorithm for each decoder. We study two extreme cases: (1) wrapping one source with the prior model given (from reference sources and model) (Section 4.1) and (2) wrapping many sources without the prior (Section 4.2). We then combine them as the general scenario (Section 4.3).

In the following, we derive the algorithm in a generic way, while also interleaving the detailed implementation, in order for readers to see how each abstract operation (such as Eq.(3) and (4)) is concretized. Since this work is to investigate the idea of source synchronization, as our implementation principle, we select reasonable implementation for the components while keeping them as simple as possible.

## 4.1 One source with the prior model

We first study how to wrap one source given the reference sources and model as the prior, which mainly exercises Principle 1, Eq.(3). This scenario often occurs in building an integration system incrementally, by adding one source at a time and constructing the wrapper for it to compromise with the existing system.

In the context, we may have reference model $\mathcal{M}_\gamma$, which specifies how some fields are generated. Meanwhile, we may have reference sources $R = \{S_a, S_b, \ldots\}$. Although they are not directly given as a "model," their wrappers $(\mathcal{W}_a, \mathcal{W}_b, \ldots)$ do generate records that can derive the underlying models (using "segmentation to model" principle which we will present in Section 4.2). Because fields in these models are correct, we combine them as a single *prior model*:

$$\mathcal{M}^p = \{f : f \in M \text{ where } M \in \{\hat{\mathcal{M}}_a, \hat{\mathcal{M}}_b, \ldots\} \cup \{\mathcal{M}_\gamma\}\}.$$

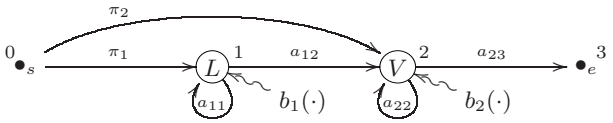



**Figure 10: Modeling a field as a 2-state HMM.**

**Implementation 1:** *How to realize each field model?* As a schema model (message) is a set of fields (message units) (Sec. 3.3.1), we need to design a scheme to model each field.

We first decide what features to use, in order to characterize the input instances. In our current design, except the literal words, we also adopt common data types, such as integer, float, month, date and time, as the features. These common data types are used across different domains and only require one-time static setup– *e.g.*, writing regular expression scripts to parse the input data and recognize the existence of each feature in the input.

For simplicity, we model a field as a 2-state HMM, based on an intuition that a field instance (*e.g.*, "Price: $20.99") usually consists of an optional label (*e.g.*, "Price:") followed by a value (*e.g.*, "$20.99"). Fig. 10 shows the model topology, where state $L$ corresponds to the label part, state $V$ is for the value, $\pi$'s are the initial probabilities, $a$'s are transition probabilities, and $b$'s are state emission probabilities. Each state represents a distribution of features (including literal words and additional features), which is specified as $b_i$. As HMM has been well-studied, we omit it here for space reason. Readers, if interested, please refer to [17] for the details. Note that any other modeling scheme, such as CRF [11], can be used instead.

Thus, our schema model is a set of field models, and each field model is a 2-state HMM. ∎

**Implementation 2:** *How can users create the reference model $\mathcal{M}_\gamma$ to assist the system?*

Since it is not intuitive for users to directly code an HMM for a field, we currently support regular-expression field patterns, *e.g.*, specifying price with "$L$[((Our|List|Best)? Price:)?] $V$[$⟨float⟩|⟨float⟩ USD]" ($L, V$ correspond to the HMM states, Fig. 10). Each field can have many patterns specified, and these patterns will then be automatically translated as field model(s) through training (Implementation 4). ∎

As Principle 1 (model to segmentation) guides, we formulate the problem of forming a new segmentation given the prior model as: For the new source $S_1$, if the prior model $\mathcal{M}^p$ is complete (*i.e.*, it contains field models for all fields in the domain or at least in the new source), find a $Y_1^* \in \mathcal{Y}_1$ most likely generated from the prior model $\mathcal{M}^p$:

$$Y_1^* = \arg\max_{Y_1' \in \mathcal{Y}_1} p(X_1, Y_1'|\mathcal{M}^p), \quad (6)$$

However, in real situations, the prior model may not be complete– *i.e.*, some fields in $S_1$ do not correspond to any model in $\mathcal{M}^p$ (*e.g.*, the reference model $\mathcal{M}_\gamma$ may only have isbn but not title and authors). In such cases, we would prefer to retain the original segmentation (produced by the base extractor) and only refine those segments strongly confirmed by the prior model. To achieve it, we thus add the current source model $\hat{\mathcal{M}}_1$ (which favors the current segmentation) and use the expected likelihood over $\hat{\mathcal{M}}_1$ and $\mathcal{M}^p$.

$$Y_1^* = \arg\max_{Y_1' \in \mathcal{Y}_1} \sum_{m \in \{\hat{\mathcal{M}}_1, \mathcal{M}^p\}} p(X_1, Y_1'|m) f(m) \quad (7)$$

where $f(m)$ is a density function used to specify the weight for each model. In our study, we fix it as a constant function– *i.e.*, the given models are treated equally.

For convenience, we will write the expected likelihood over a set of models $\mathcal{M}$, given a density function $f(\cdot)$ as:

$$\sum_{m\in\mathcal{M}} p(X_1, Y_1'|m)f(m) = E\left[p(X_1, Y_1'|m)|\mathcal{M}\right]$$

**Implementation 3:** *How to realize the consistency measure $p(X_s, Y_s'|m)$, for a given segmentation $(X_s, Y_s')$ and a schema model $m$?* This is the core of our reinforcing mechanism (Section 3.3.2), used in Eq.(7).

By assuming that the data records, *i.e.*, $(x_i, y_i')$ in $(X_s, Y_s')$, are conditionally independent given the model, we can expand the likelihood $p(X_s, Y_s'|m)$ to be:

$$p(X_s, Y_s'|m) = \sum_{x_i\in X_s, y_i'\in Y_s'} p(x_i, y_i'|m)$$

Further, we adopt the best generating field model to correspond each segment– *i.e.*, for each field segment $z_j \in (x_i, y_i')$, we select its matched field model as the $f \in m$ that has the highest conditional likelihood $p(z_j|f)$. Thus, the overall likelihood of a record $(x_i, y_i')$ given $m$ is computed as:

$$p(x_i, y_i'|m) = \sum_{z_j\in(x_i,y_i')} \max_{f\in m} p(z_j|f)$$

As $z_j$ is a sequence of tokens and $f$ is an HMM, $p(z_j|f)$ can be computed by an efficient dynamic programming technique, named Forward-Backward (see [17]). Our implementation thus follows this standard procedure. ∎

## 4.2 Many sources without the prior model

We then study the opposite scenario: How to wrap multiple sources consistently without given the prior model? This scenario, which helps us to introduce Principle 2, Eq(4), into our framework, occurs in building an integration system on a few, say 5–10, sources, from scratch.

Without given the prior model, we formulate the problem as iterative optimization of discovering models and segmentation simultaneously. The idea is to start with an initial segmentation for all peer sources (by the base extractor $\mathcal{I}$), and iteratively improve it using the models derived from the current estimate.

Suppose there are $n$ new sources $N = \{S_1, \ldots, S_n\}$. The process takes the base extraction as the initialization:

$$Y^{(0)} = \{Y_1^{(0)}, Y_2^{(0)}, \ldots\} \quad \text{where } (X_s, Y_s^{(0)}) = \mathcal{I}(S_s). \quad (8)$$

As Principle 2 guides, it then trains the models that best generate the current segmentation ($t$ initialized as 0):

$$\hat{\mathcal{M}}_s^{(t)} = \arg\max_m p(m|X_s, Y_s^{(t)}) \qquad 1 \le s \le n, \quad (9)$$

$$\hat{\mathcal{M}}^{(t)} = \{\hat{\mathcal{M}}_1^{(t)}, \ldots, \hat{\mathcal{M}}_n^{(t)}\},$$

Using Principle 1, which has been described in Section 4.1, it utilizes the current source models to form new segmentation.

$$Y_s^{(t+1)} = \arg\max_{Y_s'\in\mathcal{Y}_s} E\left[p(X_s, Y_s'|m)|\hat{\mathcal{M}}\right] \quad 1 \le s \le n \quad (10)$$

The process (Eq.(9) and (10)) repeats till stabilization.

**Implementation 4:** *How to train a source model $\hat{\mathcal{M}}_s$ given a segmentation $(X_s, Y_s)$?* This is abstracted as the optimization function of Eq.(9).

To train a schema model, we actually train a set of field models. The records $(X_s, Y_s)$ have been split into segments. Thus, we train a field model for each group of aligned segments. Training a field consists of labeling the tokens in the segments and learning the model parameters.

Labeling: Each token in each segment will be assigned as $L$ or $V$ (Fig. 10). For initialization Eq.(8), we use:

The Algorithm TS: $(N = \{S_1, \ldots, S_n\}, R = \{S_a, \ldots\}, \mathcal{M}_\gamma)$

- i-step (initialization) $t = 0$

$$(X_s, Y_s^{(0)}) = \mathcal{I}(S_s) \qquad S_s \in N$$

$$(X_r, Y_r) = \mathcal{W}_r(S_r), \quad \hat{\mathcal{M}}_r = \arg\max_m p(m|X_r, Y_r) \quad S_r \in R$$

$$\mathcal{M}^p = \hat{\mathcal{M}}_a \cup \hat{\mathcal{M}}_b \cup \cdots \cup \mathcal{M}_\gamma$$

- m-step (Principle 2: segmentation to model)

$$\hat{\mathcal{M}}_s^{(t)} = \arg\max_m p(m|X_s, Y_s^{(t)}) \qquad S_s \in N$$

$$\hat{\mathcal{M}}^{(t)} = \{\hat{\mathcal{M}}_1^{(t)}, \ldots, \hat{\mathcal{M}}_n^{(t)}\},$$

- e-step (Principle 1: model to segmentation)

$$Y_s^{(t+1)} = \arg\max_{\cancel{Y_s'\in\mathcal{Y}_s},\ Y_s'\in N(Y_s^{(t)})} E\left[p(X_s, Y_s'|m)|\hat{\mathcal{M}}^{(t)}, \mathcal{M}^p\right] \quad S_s \in N$$

The m- and e-steps are repeated until 1) the difference between two consecutive iterations is small, or 2) a pre-specified number of iterations is reached.

**Figure 11: The formal algorithm.**

*Colon-based labeling*: The part before the colon is annotated as label ($L$) and the rest as value ($V$).

When in iterations, each segment is relabeled by the field models (in $\hat{\mathcal{M}}^{(t)}$) using standard Viterbi algorithm (see [17]). Although the colon-based rule is primitive, the mislabeled tokens might get recovered later, when relabeled by the models from other sources.

Learning: The parameters ($\pi$'s, $a$'s, and $b$'s) are learned using standard *maximum likelihood estimation* (omitted here; please see [17]). Note that the data are labeled automatically (by the initialization or relabeling by other HMMs in the previous iteration). Although learning parameters is itself supervised, with auto-labeled data as input, the overall training process is unsupervised. ∎

## 4.3 Putting Together: The General Scenario

With the two extreme scenarios above, we now combine them for the general scenario of wrapping multiple sources with the prior model given. The prior model $\mathcal{M}^p$ may not be complete, while its field models are correct. If some data fields in new sources do not correspond to any field in $\mathcal{M}^p$, we would prefer they are wrapped consistently across all new sources (with models $\hat{\mathcal{M}}$). However, for data fields having corresponding models in $\mathcal{M}^p$, we would prefer they are wrapped in the same way specified in $\mathcal{M}^p$.

Therefore, the general algorithm is a combination of using the peer source models $\hat{\mathcal{M}}$ and the prior model $\mathcal{M}^p$ to perform the segmentation. Combining Eq.(7) and (10), we present the overall algorithm in Fig. 11. We emphasize that algorithm TS is essentially a combination of the dual principles (of Models ⇔ Segmentations) in iterations to refine the segmentations generated by the initial base extractors $\mathcal{I}$. In summary, TS defines the algorithm that each *Decoder* will execute, overall achieving a turbo synchronized decoding.

Since the segmentation space $\mathcal{Y}_s$ is large, directly maximizing over it is intractable. To make the algorithm practical, we further modify the e-step (Fig. 11) to search on the neighbors of the current estimate– $N(Y_s^{(t)})$.

**Implementation 5:** *How to generate candidate segmentations, instead of searching the whole solution space?*

For efficiency, we adopt the nearest neighbors of the current segmentation $N(Y_s^{(t)})$, by applying once any of the following two operations (as Fig. 7 illustrates):

- *Merge*: Given the current segmentation, *merge* attempts to merge two adjacent segments as one.
- *Split*: Given the current segmentation, *split* attempts to split one segment into two.

Theoretically, given any segmentation, we can derive any other segmentation by these two operations– for example, merging all segments as one and then splitting it into the target segments. Thus, these two operations are complete in terms of generating the solution space. This ensures that the algorithm has a chance to reach a good convergence.

If a record consists of $\ell$ tokens, there are $\ell - 1$ such neighbors, which is linear to the record length and much smaller than the whole space $\mathcal{Y}_s$. ∎

The procedure shown in Fig. 11 can be regarded as an EM algorithm [8, 6]: The i-step, preparing a solution guess, corresponds to the initialization step in EM; the m-step, estimating the distribution of the hidden variable (models), corresponds to the expectation step; and the e-step, maximizing the expectation of the likelihood, corresponds to the maximization step. For the efficiency reason ($\mathcal{Y}_s$ is too large), the e-step is modified to search on the neighbors of current estimation– $N(Y^{(t)})$. This form of the algorithm corresponds to Generalized EM (GEM) and is also guaranteed to converge, probably with the cost of more iterations [6, 16].

As a remark, we note that it is natural to arrive at an EM-like framework in our derivation. Because we resort to turbo codes as our conceptual model, while we did not explicitly follow the turbo decoding procedure, we naturally reach the same EM-style algorithmic framework, which is essentially how turbo decoders work [3].

Since the neighbors of the current estimation ($N(Y^{(t)})$) is bounded by a constant (in our observations in Section 2.2, a record has around 10 fields– thus, the neighbors of a segmentation are not many), the complexity of running the overall algorithm depends on the number of sources and the number of iterations. It is clearly linear to the number of iterations. And it is quadratic to the number of new sources, as each source is compared to all others.

Notice that our approach allows alternative implementation– for example, instead of HMM, we can use simple unigram models or more complex CRF [11, 19] to model a field. However, in this work, we focus on the turbo iterative process, rather than the individual component. Thus, the components are realized using various standard techniques.

# 5. EXPERIMENTS

Towards a more automatic integration system, we design a synchronization layer to interconnect source wrapping and matching (Fig. 1 and 2). For new online sources to be added, they are processed by the automatic extractor, and then synchronized with the existing system (new sources, reference sources and model) to produce consistent wrappers and thus facilitate the subsequent matching to integrate the wrappers (*i.e.*, aligned with the wrappers of other sources).

Accordingly, we evaluate: First, whether assembling with the existing automatic extractors, our system can indeed reduce the inconsistency errors (Section 5.1). Second, whether the outputs can facilitate the follow-up matcher to produce better integrated results (Section 5.2).

To setup the experiments, we collected the data from three domains: albums, books, and cars. For every domain, we collected 10 sources from the deep-Web. They are all popular sources– for example, book sources included famous amazon.com and bn.com. For each source, we submitted three or four queries. Example queries were {artist="Enya"} for albums and {title="Harry Potter"} for books. For each query, we collected the first response list page, which roughly contained 10 records. Hence, for each domain with 10 sources, there were 30–40 sample pages, containing 300–400 records in them. We thus used this data set for experiments.

To run the experiments, we considered integrating sources from scratch: We wrapped all sources together using an unsupervised setting (without references). During evaluation, we corrected the errors. We also examined the data, particularly those producing errors, and created assistant rules as the reference model. Later, we used these correct data, and the user-crafted model, to run a supervised setting (with references). This scenario reflected a practical situation of accumulating an integration system with more and more sources and assistant knowledge.

## 5.1 Extraction

First, we did experiments to examine how our system can reduce the extraction inconsistency errors.

**Setup:**
We adopted 3 automatic Web data extractors available in the literature– RoadRunner(RR) [5], ExAlg(EA) [2], and TreeAlign(TA) [22]. Because our ultimate goal is automatic integration, we consider only the unsupervised automatic extractors, which need no tuning (or training) for the domain– *e.g.*, some traditional IE methods, like HMM and CRF, are not in our consideration.

While the three extractors all involve sophisticated algorithms to analyze the input pages, we would like to examine whether our approach can benefit "weak" extractors. Therefore, we design a simple segmentation rule that only uses line break to segment the fields:

*Line-based splitting*: A record is splitted into fields by *line breaks* (caused by tags <br>, <p>, <tr>, etc).

Let's name this as LineSplitter (LL). It is undoubted that this rule is simple, and thus enables us to see whether our approach can benefit such a weak extraction approach.

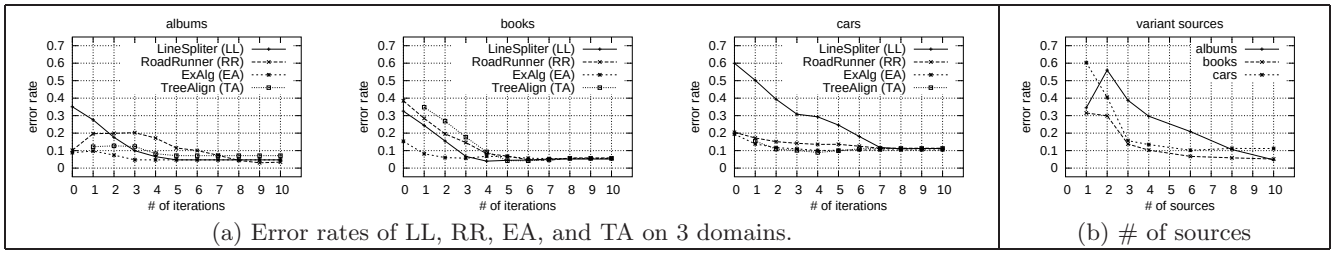
In Section 2, we have already shown that these automatic extractors did not perform well for the three testing domains. In fact, from that experiment, we observe that books is a hard domain (full of variations) for most existing methods; albums and cars are moderate, while different methods work more uniformly in cars but still divergently in albums (see Fig. 4).

By using different extractors with different situations from the testing domains, we can examine whether our framework is robust enough to cope with these divergent settings.

To measure the accuracy, we count the errors in every output record and report the error rate– the average percentage that a field is extracted incorrectly in a record.

**Unsupervised Setting:**
We first ran the experiments using an unsupervised setting: For each domain, we used all 10 sources as the input (without any reference source and model) and recorded the intermediate results of each iteration. The extraction output generated by the existing extractors was recorded as iteration 0 (meaning the results before applying our approach). Fig. 12a depicts, for every domain and every extractor, the error rate at each iteration.

To analyze the performance and behavior of our frame-




(a) Error rates of LL, RR, EA, and TA on 3 domains. (b) # of sources

**Figure 12: Extraction performance in the unsupervised setting.**

work, we examine the following factors: the cooperating extractors (LL, RR, EA, TA), the iterations, and the sources.

*Extractors*: To examine the results of the 4 cooperating extractors (LL, RR, EA, TA), we first look at the error rates they produced (iteration 0). From the curves, the original performance of the existing extractors (RR, EA, TA) are generally better than that of LL. For the 3 domains, RR performs well in two of them (a:.10, c:.20 but b:.38); EA performs well for three (a:.09, b:.15, c:.19); TA performs moderately (a:.12,b:.35,c:.15); and LL performs relatively worse (a:.35,b:.33,c:.60). This confirms that the public extractors are indeed more sophisticated than the simple rule LL.

However, while comparing the final performance of adding our framework (the last iteration), the results of the 4 extractors are comparable; the error rate declines from a:.09-.35. to .03-.07, b:.15-.38 to .05-.06, and c:.15-.60 to .10-.11. This phenomenon implies that the performance of applying our framework converges, independently on the existing extractors invoked. And the final performance is indeed better than that of each individual extractor, implying that the source synchronization does help.

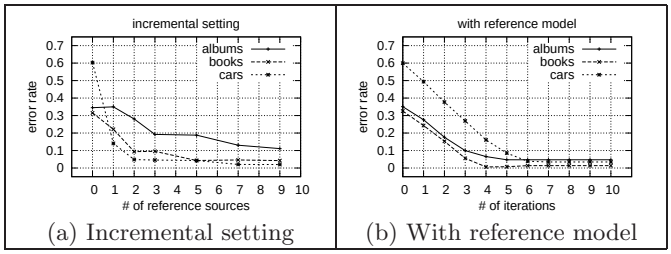
*Iterations*: In addition, the curves in Fig. 12a indicate that the performance improves as the iterations increase. This is most clear when the algorithm starts with a weak extractor (*e.g.*, LL). An exception is the curve for RR on albums (the 1st curve in Fig. 12a), which rises at early iterations and drops later. The reason is that a few poorly-extracted sources may have misled the algorithm in early stages. But, it is interesting to see that, as more iterations are performed, the poor sources get extracted better and better, and the performance gets recovered. This phenomenon indicates that our framework is rather robust.

*Sources:* We further examine how the number of sources relates to the performance. For each domain, from number 1 to 10, we randomly picked up an equal number of sources as the input and recorded the final performance. The process was performed 30 runs for each number. Fig. 12b shows the average performance.

The curves in Fig. 12b show that in general, the performance improves as the sources increase. This is extremely encouraging as it indicates that our approach can indeed leverage multiple sources to improve the performance on individual sources. We notice an exception for albums, where a sharp drop occurs at the 2-source case. It is because for two sources, the reinforcing becomes tied– an error affects a correct extraction and vice versa, with equal effect. However, the performance quickly recovers and continues to improve as we have more sources available.

**Supervised Setting:**

With the extracted data from the unsupervised setting, we corrected errors to obtain correct extraction for each source. We then used this data set to evaluate our framework on utilizing the previous extracted sources (as reference sources)




(a) Incremental setting (b) With reference model

**Figure 13: Extraction in the supervised setting.**

**Table 1: Comparison of error rates.**

| | unsupervised setting | | | | supervised | |
|---|---|---|---|---|---|---|
| domain | LL | RR | EA | TA | ref. sources | ref. model |
| albums | **.35↘.05** | .10↘.03 | .09↘.06 | .12↘.07 | .11 | .05 |
| books | **.33↘.05** | **.38↘.06** | .15↘.05 | **.35↘.06** | .04 | .01 |
| cars | **.60↘.11** | .20↘.11 | .19↘.10 | .15↘.11 | .02 | .03 |

(The cells with improvement greater than 20% are highlighted.)

or the accumulated reference model to help the processing of newly added sources.

*Reference sources.* We simulated an incremental wrapping process by adding sources one by one until all 10 sources are used up. At each step, a new source was randomly picked up and added into the system with all previously-wrapped sources used as the references. The performance of extracting the new source was recorded. Then, the correct extraction of the new source was incorporated into the collection of reference sources, as the prior for the next run. For each domain, we repeated the whole process 30 times.

Fig. 13a depicts the result, where the x-axis is the number of reference sources involved and the y-axis denotes the average error rate. The curves show, as we would expect, that the performance consistently increases as we have more reference sources to work on. In addition, comparing Fig. 13a and 12b, it is clear, perhaps not surprising, that the performance converges quicklier using reference sources.

*Reference model.* We lastly examine how the framework can cooperate with users to further improve the process. As we have seen, even when the wrapped sources are added, the performance is still not perfect. In an operational environment, we could let a user examine the results and possibly create templates for problematic fields– *e.g.*, price may be supplemented by template "$⟨float⟩" and authors "Authors: ⟨word⟩+." We added (a:1, b:7, c:5) such manually-crafted field templates to be the reference model.

Fig. 13b shows the performance with LL as the base extractor. The final performance is better than that without reference model. Also, it reaches convergence quicklier (*i.e.*, with fewer iterations). To summarize, we lists all the error rates in Table 1. For example, the unsupervised error rate of using RR in books is .38 and the final is reduced to .06; it achieves .04 using reference sources and .01 by adding reference model. Generally, importing user-crafted model can raise the performance.

In summary, our approach indeed can cooperate with dif-

ferent extractors in different domains. It is robust, as achieving constantly good performance in different situations, even starting with a weak extractor (like LL). Moreover, the spirit of synchronization is indeed in action: The more sources it handles (including reference sources and model), or the more iterations (before convergence), the better the results are.

## 5.2 Matching

Then, we examine whether the new output can facilitate matching to produce a better final result.

**Setup:**
Since our integration considers multiple sources, we design several multi-source matching approaches (there is not much work in this direction– most existing approaches are pairwise, *i.e.*, consider merely two sources). First, we adopted clustering [21]; let's name it ClusMatch: Given all extracted fields, ClusMatch first treats each field as a singleton cluster; then it merges the two most similar clusters as a new one iteratively until a pre-specified cluster number is reached. The fields grouped in the same cluster are considered matched. Note that clustering naturally supports complex matching.

Second, we extended pairwise matching to work in a multi-source manner; let's name it ProgMatch: Given a sequence of sources (*e.g.*, 1-2-3-4), ProgMatch matches the first two sources (*e.g.*, 1-2), and "progressively" adds a source into the current matching, one by one (*e.g.*, becoming (((1-2)-3)-4)); in each step, it uses the pairwise matching [18].

Third, we applied human to do best matching (like Example 1); let's name it BestMatch: Given all extracted fields, we manually identify and match those fields with overlapped semantics. Note that BestMatch just reacts to the given input passively; it does not modify any input field– any compound field will remain unchanged.

By using these different matching approaches, we can examine whether our framework is generally beneficial for any reasonable matching approaches.

To measure the performance, we compare the result with the manually-prepared integrated tables. The matching accuracy is measured using F-measure [15, 21]– a combination of precision (how many matched field pairs are correct) and recall (how many correct matched pairs are identified). Let $a$ be the set of matched field pairs produced by testing matcher and $b$ be the set of correct matched pairs; then the precision $P$, recall $R$, and F-measure $F$ are defined as

$$P = \frac{|a \cap b|}{|a|}; \quad R = \frac{|a \cap b|}{|b|}; \quad F = \frac{2RP}{R+P}.$$

**Results and Analysis:**
Following the extraction experiments (Sec. 5.1), we applied the testing matchers on every extraction result we recorded (for each extractor on each domain at each iteration). Fig. 14 summarizes the matching performance in a 3x3 grid, each row for a domain, each column for a matcher, and each cell plotting the F-measure for the matching obtained on the extraction output of each extractor at each iteration.

The curves clearly show the trend of performance improvement and convergence. To reveal it, we look at the perspectives: the extractors and matchers.

*Extractors*: Looking at each subfigure in Fig. 14, we observe that the initial performances (of matching on the extraction output produced purely by the cooperating extractors) are quite divergent. For example, in books+ClusMatch, the range of initial performances (iteration 0) is .35-.66. However, the final performances (the last iteration) generally converge to a better and much narrower range– *e.g.*, .82-.87 in books+ClusMatch. This indicates that incorporating our framework into matching indeed produces a better and stabler matching output, regardless of the extractors.

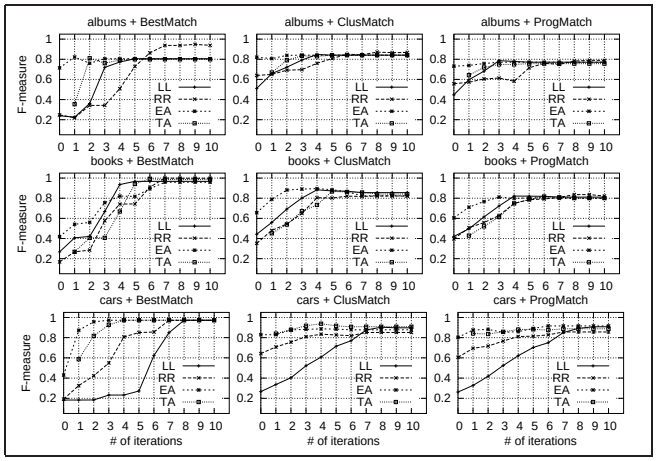



**Figure 14: Matching performance.**

**Table 2: Matching performance improvement.**

| albums | LL | RR | EA | TA |
|---|---|---|---|---|
| BestMatch | **.24↗.81** | **.25↗.94** | .71↗.81 | **.35↗.80** |
| ClusMatch | **.51↗.84** | **.64↗.87** | .82↗.84 | .67↗.84 |
| ProgMatch | **.45↗.77** | **.56↗.79** | .73↗.77 | .64↗.76 |
| **books** | **LL** | **RR** | **EA** | **TA** |
| BestMatch | **.27↗.97** | **.17↗.99** | **.41↗.96** | **.27↗.99** |
| ClusMatch | **.44↗.86** | **.35↗.82** | .66↗.82 | **.45↗.84** |
| ProgMatch | **.42↗.81** | **.40↗.82** | **.60↗.80** | **.43↗.80** |
| **cars** | **LL** | **RR** | **EA** | **TA** |
| BestMatch | **.18↗.97** | **.19↗.97** | **.43↗.97** | **.59↗.97** |
| ClusMatch | **.27↗.90** | **.64↗.85** | .83↗.88 | .84↗.91 |
| ProgMatch | **.26↗.91** | **.61↗.86** | .80↗.92 | .84↗.89 |

(The cells with improvement greater than 20% are highlighted.)

*Matchers*: While different matchers have different capabilities (*e.g.*, they use different techniques and might achieve different performances), from the curves in Fig. 14, they all behave in a similar way: Their performances improve as the iterations increase. From our previous experiments of extraction, we have observed that the extraction improves as the iterations increase. This, perhaps not surprising, implies that extraction and matching are positively correlated– *i.e.*, matching can get improved better when fed with a more consistent extraction output.

We provide a summary table showing the matching performances for various combinations of domains, extractors, and matchers (see Table 2). In each cell, we show two numbers, which indicate the results before and after our framework is applied, respectively. From the table, it clearly shows that the matching performances get improved, by adding on our framework to any extractor and any matcher.

In summary, for different domains and different matchers, our framework can provide more feasible extraction outputs for matching to produce better and stabler final matching results, regardless of the cooperating extractors. This indeed confirms our pursuit of synchronization to boost the overall integration.

# 6. RELATED WORK

Wrapper construction, and also its broader area of data extraction from the Web, is crucial for integration systems of online sources and has been studied actively [10]. However, most works address it as a standalone extraction task on a

single source, ignoring its contexts in an integration system. Below, we briefly relate and contrast our work.

There are many works on extracting data from Web pages, in different settings–manual, semi-automatic, and automatic (see [10]). For manual and semi-automatic methods (*e.g.*, [7, 20]), data fields are identified explicitly by human (programming wrappers or labeling data to train wrappers). Thus, if fields are identified or labeled consistently across sources, the constructed wrappers have no inconsistency issue. However, as they need "manual" efforts, these methods are hardly scalable and also unsuitable for on-demand scenarios [23].

For automatic methods (*e.g.*, [2, 5]), they identify data fields using various regularities, *e.g.*, the underlying page templates (*e.g.*, delimiters [9], tag tree [22], visual clues [23]) and site structure (*e.g.*, detailed pages [12]). While these methods perform sophisticated page- or site-level analyses, however, they are not context-aware– *i.e.*, their output among different sources may not be consistent (as we have shown). By leveraging these existing methods (with their sophisticated syntax analyses), on top of them, our approach explicitly considers the context-awareness of integration by synchronizing the extracted content among sources.

While wrappers capture the page template and are highly source-specific, there is a large and active area of information extraction (IE) which focuses on unstructured text [4]. Their techniques are mostly statistical and focus more on the content model, which will be valid across different format of similar data and thus not source-specific. However, they are not currently suitable for wrapper generation, because they do not take the underlying page structure to generate template for nearly-perfect and efficient extraction. In addition, we stress that, with our synchronized extraction, we are in fact attempting to combine the "template"-strength of current wrappers with the content synchronization across sources, therefore a marriage of both merits.

Recently, several works also share the similar insight of using certain domain models or existing data in guiding extraction, and therefore apply more IE-based approaches. First, to use domain models, [24] applies IE by training statistical models like HMM or CRF. Second, to leverage existing data, [1, 14] convert the clean reference table(s) as extraction models. In comparison, our work aims at a broader notion of "context," which accounts for not only prior knowledge (domain models and existing wrappers) but also current peer sources. That is, our framework must handle, in an "unsupervised" way, dynamic alignment between current sources and prior models, while these existing techniques "pre-train" a model and do not online adapt.

To enable data exchange and integration, while avoid difficult extraction, sources may provide Web services, such as amazon.com and google.com, to deliver their data in a more structured form, using XML. However, most current online sources still present their results in legacy HTML formats. Moreover, even the data are XML-tagged, they may not be tagged in the right granularity of their semantics to compromise with the integration system– *e.g.*, a source tags publish date as a single field ⟨pub-date⟩ while another tags it as two fields ⟨month⟩ and ⟨year⟩. Thus integrating such structured data still needs consistent re-extraction on them.

Also, we focus on the spatial context (*i.e.*, peer sources, reference sources, etc). The temporal setting, such as wrapper maintenance [13], in terms of our framework, can be immediately regarded as temporal context using cached source models to recognize format-changed sources.

# 7. CONCLUSIONS

In this paper, we propose synchronized extraction to fundamentally support context-aware wrapper construction for integration systems. To realize it, we develop Turbo Syncer, which incorporates existing automatic extraction techniques, and on top of them, adds content synchronization, therefore to directly benefit the follow-up matching task. The extensive experiments show the promising of the framework.

# 8. REFERENCES


[1] E. Agichtein and V. Ganti. Mining reference tables for automatic text segmentation. In *Proc. of SIGKDD*, 2004.

[2] A. Arasu and H. Garcia-Molina. Extracting structured data from web pages. In *Proc. of SIGMOD*, 2003.

[3] C. Berrou, A. Glavieux, and P. Thitimajshima. Near shannon limit error-correcting coding and decoding: Turbo codes. In *Proc. of IEEE Int. Conf. on Commun.*, pages 1064–70, 1993.

[4] J. Cowie and W. Lehnert. Information extraction. *Communications of ACM*, 39(1):80–91, 1996.

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

[6] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39:1–38, 1997.

[7] J. Hammer, J. McHugh, and H. Garcia-Molina. Semistructured data: the TSIMMIS experience. In *Proc. of the 1st East-Euro Symp. on Advances in DB and Info. Syst.*, 1997.

[8] H. Hartley. Maximum likelihood estimation from incomplete data. *Biometrics*, 14:174–194, 1958.

[9] N. Kushmerick, D. S. Weld, and R. B. Doorenbos. Wrapper induction for information extraction. In *Proc. of IJCAI*, 1997.

[10] A. Laender, B. Ribeiro-Neto, A. Silva, and J. Teixeira. A brief survey of web data extraction tools. *SIGMOD Record*, 31(2):84–93, June 2002.

[11] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmentating and labeling sequence data. In *Proc. of ICML*, 2000.

[12] K. Lerman, L. Getoor, S. Minton, and C. Knoblock. Using the structure of Web sites for automatic segmentation of tables. In *Proc. of SIGMOD*, 2004.

[13] K. Lerman, S. N. Minton, and C. A. Knoblock. Wrapper maintenance: A machine learning approach. *Journal of Artificial Intelligence Research*, 18:149–181, 2002.

[14] I. R. Mansuri and S. Sarawagi. Integrating unstructured data into relational databases. In *Proc. of ICDE*, 2006.

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

[16] R. M. Neal and G. E. Hinton. A view of the EM algorithm that justifies incremental, sparse, and other variants. *Learning in Graphical Models*, pages 355–368, 1998.

[17] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[18] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4):334–350, 2001.

[19] S. Sarawagi and W. W. Cohen. Semi-markov conditional random fields for information extraction. In *Proc. of NIPS*, 2004.

[20] S. Soderland. Learning information extraction rules for semi-structured and free text. *Machine Learning*, 34(1-3):233–272, 1999.

[21] W. Wu, C. Yu, A. Doan, and W. Meng. An interactive clustering-based approach to integrating source query interfaces on the deep web. In *Proc. of SIGMOD*, 2004.

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

[23] H. Zhao, W. Meng, Z. Wu, V. Raghavan, and C. Yu. Fully automatic wrapper generation for search engines. In *Proc. of WWW*, 2005.

[24] J. Zhu, Z. Nie, J.-R. Wen, B. Zhang, and W.-Y. Ma. 2d conditional random fields for Web information extraction. In *Proc. of ICML*, 2005.