A Semiautomatic Approach for Accurate and Low-Effort Spreadsheet Data Extraction

Zhe Chen
University of Michigan
Ann Arbor, MI 48109-2121
chenzhe@umich.edu

Michael Cafarella
University of Michigan
Ann Arbor, MI 48109-2121
michjc@umich.edu

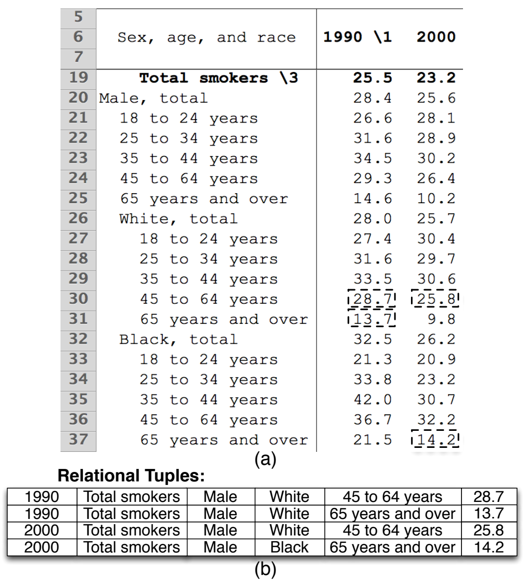
ABSTRACT

Spreadsheets contain valuable data on many topics, but they are difficult to integrate with other sources. Converting spreadsheet data to the relational model would allow relational integration tools to be used, but using manual methods to do this requires large amounts of work for each integration candidate. Automatic data extraction would be useful but it is very challenging: spreadsheet designs generally requires human knowledge to understand the metadata being described. Even if it is possible to obtain this metadata information automatically, a single mistake can yield an output relation with a huge number of incorrect tuples.

We propose a two-phase *semiautomatic* system that extracts accurate relational metadata while minimizing user effort. Based on conditional random fields (CRFs), our system enables downstream spreadsheet integration applications. First, the *automatic extractor* uses hints from spreadsheets' graphical style and recovered metadata to extract the spreadsheet data as accurately as possible. Second, the *interactive repair* component identifies similar regions in distinct spreadsheets scattered across large spreadsheet corpora, allowing a user's single manual repair to be amortized over many possible extraction errors. Through our method of integrating the repair workflow into the extraction system, a human can obtain the accurate extraction with just 31% of the manual operations required by a standard classification based technique. We demonstrate and evaluate our system using two corpora: more than 1,000 spreadsheets published by the US government and more than 400,000 spreadsheets downloaded from the Web.

1. INTRODUCTION

Spreadsheets are a critical data management tool in many fields [3]. In many of these cases, the data fits the relational model, but the user has chosen to use spreadsheet software instead of a relational database: spreadsheets are effective at performing selection, projection, sorting, and sometimes even Extract-Transform-Load (ETL). However, there is at least one area where spreadsheet functionality badly trails the relational world: data integration. This is a shame; given that spreadsheet-embedded data is diverse and of high value, many analysis-driven questions should be answerable with a good *ad-hoc* spreadsheet integration tool.



5			
6	Sex, age, and race	1990 \1	2000
7			
19	**Total smokers \3**	**25.5**	**23.2**
20	Male, total	28.4	25.6
21	18 to 24 years	26.6	28.1
22	25 to 34 years	31.6	28.9
23	35 to 44 years	34.5	30.2
24	45 to 64 years	29.3	26.4
25	65 years and over	14.6	10.2
26	White, total	28.0	25.7
27	18 to 24 years	27.4	30.4
28	25 to 34 years	31.6	29.7
29	35 to 44 years	33.5	30.6
30	45 to 64 years	28.7	25.8
31	65 years and over	13.7	9.8
32	Black, total	32.5	26.2
33	18 to 24 years	21.3	20.9
34	25 to 34 years	33.8	23.2
35	35 to 44 years	42.0	30.7
36	45 to 64 years	36.7	32.2
37	65 years and over	21.5	14.2

(a)

Relational Tuples:

1990	Total smokers	Male	White	45 to 64 years	28.7
1990	Total smokers	Male	White	65 years and over	13.7
2000	Total smokers	Male	White	45 to 64 years	25.8
2000	Total smokers	Male	Black	65 years and over	14.2

(b)

Figure 1: A spreadsheet about smoking rates, from the Statistical Abstract of the United States.

If spreadsheet data could be easily converted to the relational model, it could benefit from society's huge investment in relational integration tools. There has been a substantial amount of work in converting grid-style data to the relational model, much of it in connection to Web HTML tables, rather than spreadsheets [1, 2, 5, 12, 15, 36, 20]. However, projects to date have either been extremely labor-intensive (and thus inappropriate for ad-hoc integration), or they have ignored data layouts that are very typical of spreadsheets.

For example, Figure 1(a) shows a portion of a spreadsheet downloaded from the government's Statistical Abstract of the United States.[1] A human reader can easily tell that the *data* value 28.7 is annotated by the *annotations* 1990, Male, White, and 45 to 64 years. We call this implicit relationship between annotations and data a *mapping*. By repeatedly finding such mappings, a human could eventually reconstruct the *relational tuples* seen in Figure 1(b).

[1] http://www.census.gov/compendia/statab/2012/tables/12s0204.xls

System Goal — The goal of this paper is to convert spreadsheets to relational tuples by solving a critical problem: recovering the *implicit mapping* between annotations and data. This mapping structure exists in many kinds of tabular data (*e.g.*, Web HTML tables and textual financial reports), but is especially common in spreadsheets. We manually examined 200 randomly selected spreadsheets from the Web and found that more than 32% of the spreadsheets in a general English-language Web crawl contain an implicit mapping between annotations and data. When examining the top ten Internet domains that publish the greatest number of spreadsheets, more than 60% of spreadsheets do so [8].

Our vision for using spreadsheet-derived relations in an ad-hoc integration tool requires that we extract the annotation-to-data mappings ***accurately*** and with ***low effort***. Finding all *accurate* mappings is important in avoiding incoherent tuples that violate the data's implicit relational schema. For example, a naïve extractor might find that 13.7 in Figure 1(a) is annotated by 65 years and older, but it must also find White and Male in order to be correct. Using incorrect tuples in the downstream integration tool can lead to misleading results in the final output. Finding mappings with *low effort* is also important because our integration tool is intended for fast ad-hoc tasks, in which the user quickly hypothesizes and examines many possible integrations. To the extent that there is prevailing practice for spreadsheets, it requires users to provide explicit extraction rules or to write custom code. But users often do not want to spend much time checking out a hypothesis that may not be proven true. Any time-consuming step can be a serious burden.

Technical Challenges — Unfortunately, simple automatic extraction mechanisms face a number of challenges. First, the annotation relationship shown in Figure 1 is clear to a human because of the textual formatting, but many other spreadsheets use different or contradictory formatting; methods based on formatting heuristics will be quite poor at reconstructing these relationships. Second, many annotation-to-data mappings implicitly rely on human understanding of domain-specific metadata. Consider a spreadsheet of US states that does not use any stylistic cues to distinguish *Michigan* from *Midwest*; the human reader's domain knowledge is what makes the annotations recoverable.

Moreover, the mapping extraction task is brittle: even a single mistake in this annotation-to-data mapping can yield an extracted relation that has many incorrect tuples, and thus ruins our data integration application. For example, if the extractor fails to recognize that Male annotates White, the system would generate incorrect relational tuples for all the data values from rows 26 to 31, even if it is otherwise entirely correct. Unfortunately, even a high-quality automatic extractor will eventually make a mistake; in order to avoid errors, a human being will have to review the extractor's output and tediously fix the extraction errors, one after another. In order to minimize work for the human, we seemingly need an extractor that is both domain-independent and produces very few errors.

Our Approach – We propose a two-phase *semiautomatic* approach to recovering spreadsheet annotation-to-data mappings accurately and with little human effort.

First, the *automatic extractor* component receives spreadsheets as input and computes a mapping without any human interaction. Based on conditional random fields [26] (CRFs), it exploits single-spreadsheet graphical style hints, such as the font and typographic alignment, that are obvious to a human observer. It also identifies and exploits correlated extraction decisions; these correlated decisions can appear within one spreadsheet or between two unrelated spreadsheets. Our resulting *automatic extractor* obtains accuracy that beats a baseline approach by up to 91% on a large workload of spreadsheets.

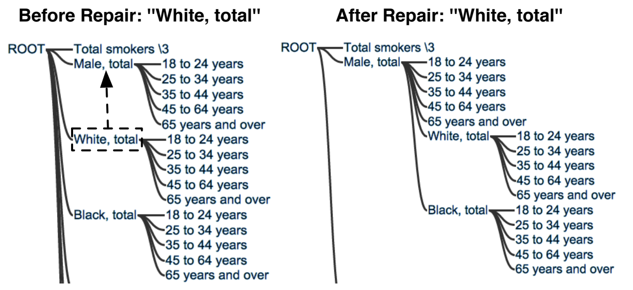


Figure 2: Our user interface for applying repairs on annotation mappings recovered from Figure 1. The *automatic extractor* recovers the annotation mappings from Figure 1, with some mistakes on the left. After the user repairs one error, *interactive repair* uses the repair to automatically fix several others, yielding the mapping on the right.

Second, our system offers an *interactive repair* phase, in which a human repeatedly reviews and corrects the *automatic extractor*'s output until no errors remain. Because our application is sensitive to errors, we expect a human will review the *automatic extractor*'s output. But our *interactive repair* system is more than simply asking a human to fix mistakes. We again exploit the correlations among different extraction decisions to make more effective use of each user repair operation. A user's single repair can be silently and probabilistically applied to multiple possible errors, allowing us to *amortize* the user's effort over many likely extractor mistakes. Building a model that can perform this amortization, and managing the inadvertent errors that such an approach might introduce (a problem we call *backtracking*), is one of this paper's core contributions.

Figure 2 shows an example of the user interface for applying repairs. The left side of the diagram indicates the initial hierarchy obtained by the *automatic extractor*. The dashed arrow shows that a user performs a repair by clicking and dragging White so that it becomes a child of Male, indicating that Male annotates White. This one repair operation triggers multiple error fixes, including setting Male to also annotate Black. By making our system part of the user's natural review-and-repair loop, we can reduce the number of manual repairs by up to 71% when compared to our already-effective *automatic extractor*.

A critical component of both the *automatic extractor* and *interactive repair* is the detection of similar extraction decisions. By automatically constructing our own domain-specific metadata resource, we can more effectively detect these decisions than when using no metadata or when using an off-the-shelf resource such as Freebase [4].

Background – Recently, several projects [2, 5, 15, 20] have tried to extract data from tabular data; however, so far, all have been labor-intensive or do not apply to the problem of finding annotation mappings for tuple reconstruction [1, 2, 12, 20]. Our solution's design, which alternates automatic and human-driven effort, is similar in spirit to Wrangler [16, 23], as well as programming-by-demonstration and mixed initiative systems [19, 21, 22, 33]. However, Wrangler-style techniques cannot be applied to our situation directly, as they generally process data with standard textual cues that are often missing from real-world spreadsheets.

In our previous work, we described the Web spreadsheet corpus and a basic form of the extractor [8]; we compared against and substantially beat that basic extractor. We also demonstrated the ad hoc integration application [9]. We have not previously described the extraction and repair technique that forms our core technical contribution. All of this paper's technical content is new, except for a small portion of the data model in Section 2 that appeared previously.

Contributions and Outline — In this paper, we focus on the problem of extracting the mapping between annotations and data. We are the first to present the semiautomatic extraction approach. Our contributions include:

- An effective CRF-based *automatic extractor* for obtaining spreadsheet annotation mappings. (Section 3.)
- An extended version of the CRF to implement *interactive repair*. This technique amortizes each user repair over many extraction errors at once. (Section 4.)
- A method for building a *domain-specific* metadata resource that assists with the above steps. (Section 5.)
- An evaluation of our overall extraction system on two distinct spreadsheet corpora. (Section 6.)

We define the relational tuple construction task in Section 2, and we cover related work in Section 7. Finally, we conclude with a discussion of future work in Section 8.

2. PRELIMINARIES

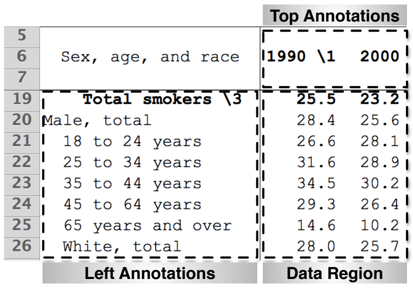
Our design goals are motivated by observations on two large spreadsheet corpora. The first is the Statistical Abstract of the United States (SAUS), published by the U.S. Census Bureau.[2] It contains relatively clean high-quality data on many topics. The second is a Web-derived collection (WEB) of 410,554 Microsoft Excel files.[3]. Its topics and authors are much more diverse than SAUS. We created the WEB corpus by looking for Excel-style file endings among roughly ten billion URLs in the English-language ClueWeb09 Web crawl [10]. In this section, we formally define the implicit mappings recovery task. We also describe some design considerations for possible solutions.

2.1 Data Model and Problem Statement

In its most generic incarnation, a spreadsheet is simply an $M \times N$ grid of cells, in which each cell can contain a string, a number, or nothing. In practice, most spreadsheets, especially the high-quality ones that carry data that we want to extract, have substantially more structures. We make two assumptions about the spreadsheets we will process without seriously compromising our approach's generality.

[2]Downloadable://www.eecs.umich.edu/˜michjc/data/saus.tar.gz

[3]http://www.eecs.umich.edu/˜michjc/data/sheets_url.txt.gz


Top Annotations
Left Annotations
Data Region

5			
6	Sex, age, and race	1990 \1	2000
7			
19	Total smokers \3	25.5	23.2
20	Male, total	28.4	25.6
21	18 to 24 years	26.6	28.1
22	25 to 34 years	31.6	28.9
23	35 to 44 years	34.5	30.2
24	45 to 64 years	29.3	26.4
25	65 years and over	14.6	10.2
26	White, total	28.0	25.7

Figure 3: The three primary components of a *data frame* spreadsheet.

Data Frames – First, we focus on a prototypical form of spreadsheet that we call a **data frame**. Figure 3 shows the three components that make up a data frame: two rectangular *annotation regions* (*left* and *top*) and a single rectangular *data region*. For each data in the *data region*, there is usually at least one *annotation* in the *top* and *left* annotation regions. For example, in Figure 3, the data 14.6 has annotations 65 years and over, Male and 1990. We previously addressed the problem of finding data frames in spreadsheets [8].

Of course, data frames are not entirely self-contained objects. Some metadata may exist across spreadsheets, either explicitly (in the form of cross-sheet formulae) or implicitly (in the form of textual annotations and user instructions). We currently only handle single-sheet data frames, and do not exploit the information across spreadsheets.

Hierarchies – Second, we focus on **hierarchical** spreadsheets. We assume a spreadsheet is *hierarchical* if the annotations in the *top* or *left* annotation region exhibit a hierarchical tree structure of at least two layers. Each annotation region has a notional tree that characterizes how each annotation describes the *data region*. In particular:

- There is a virtual *root* node for *left* (*top*).
- Each annotation corresponds to a node in a tree. There is at least one tree in *left* and at least one in *top*.
- Each annotation in *left* (*top*) *directly* annotates the row (column) of the data to the right (below).
- Each annotation also *indirectly* annotates the data that are *directly* annotated by its descendants.

For example, in Figure 1, all the data at row 31 have a *direct* annotation of 65 years and older and *indirect* annotations of White, Male and Total smokers in the *left* hierarchy. Note that the annotation hierarchies are *not* ontologies. For example, Male is not a super-category of its children.

It may appear that hierarchical relationships describe data aggregations. For example, set of Male smokers should be the sum of its subcategories. However, in practice clear aggregation relationships are surprisingly rare. Even in cases where it appears that an aggregation relationship should exist, there may be hidden weights on components in a sum (such as the size of each demographic group) or implicit filters that modify the aggregate (such as removing Puerto Rico's population from the "Total US" population). If our system limited itself to unambiguous aggregations, it would recover very little of the annotation mapping.

Spreadsheets that exhibit both **data frames** and **hierarchies** are very widespread. We randomly chose 200 samples from our Web crawled spreadsheets and manually marked their structures. We found 50.5% of the spreadsheets consist of *data frame* components, and 32.5% have hierarchical top or left annotations. In this paper, we primarily discuss *left* annotation hierarchies, but hierarchies also exist in *top*. However, *top* annotation hierarchies are generally easier to recover than *left*, as the row and column number in *top* are very strong and reliable clues [8].

Relational Tuples — Given the annotation hierarchies for a spreadsheet, we can recover the equivalent relational tuples. For each data in the data region, we generate a relational tuple that consists of the following: (1) The data itself; and (2) Its direct annotation plus all of its indirect annotations in the *left* and *top*. For example, in Figure 1, the data 28.7 has a direct annotation of 45 to 64 years and indirected annotations of White, Male and Total smokers in *left*. Similarly, we obtain the direct and indirected annotations in the *top* hierarchy and then generate the relational tuple as shown in Figure 1 (b).

Combining multiple relational *tuples* into a single relational *table* is sometimes straightforward, but may depend on data-specific details to align each value of a tuple properly to a consistent attribute. Elmeleegy, *et al.* [14] proposed a relevant method to perform this alignment in the context of relational table extraction from the Web.

Problem Statement — Thus, extracting the annotation hierarchies is core to our implicit mappings recovery task. We now describe the annotation hierarchy more precisely.

Let $A = \{a_1, ..., a_N, root\}$ be a set of annotations in an annotation region, where *root* is a synthetic node as the root of every hierarchical tree. Given $a_p, a_c \in A$, we say (a_p, a_c) is a *ParentChild* pair if a_p is the parent of a_c in the annotation hierarchy. For example, in Figure 1, *(row-20, row-26)* (the strings (Male, White)) is a *ParentChild* pair, while *(row-25, row-26)* (the strings (65 years and over, White)) is not. We now formally define the *annotation hierarchy*.

Definition 1. *(Annotation Hierarchy) We define $H = (A, P)$ to be an* **annotation hierarchy** *for annotations A in an annotation region, where $P = \{ParentChild(a_i, a_j) \mid a_i, a_j \in A\}$ is the set of all the ParentChild pairs in A.*

The spreadsheet annotation-to-data mapping task, thus, amounts to recovering the *ParentChild* pairs *P* for its *left* and *top* metadata regions. We also call this the *hierarchy extraction* task. For example in Figure 1, the solution for *left* contains the *ParentChild* pairs {*(19, 20), ..., (32, 37)*}.

2.2 Design Discussion

We can now describe our design choices behind the *automatic extraction* and our mechanism for exploiting user information during *interactive repair*.

Automatic Extraction — A ***basic*** method for solving the *hierarchy extraction* problem uses only local alignment and indentation information. Using these simple features, this method classifies two annotations as having a *ParentChild* relationship or not and assigns the most probable parent annotation to each child. This mechanism may work on very well-formatted annotation regions as in Figure 1, but in general will perform poorly: spreadsheets describe hierarchies using many different and conflicting graphical styles.

We can improve this method by incorporating a wider range of features that are local to the *ParentChild* pair, such as counting how many lines separate two annotations, or testing whether a word contains "total" (thus indicating a likely parent). We can also enforce certain quality constraints on the resulting hierarchy structure, by requiring that the overall result have a single root and no cycles. However, this extractor still has fairly limited ambitions: it makes little attempt to exploit the actual annotation decision correlations in the corpus.

Many state-of-the-art document extractors use joint inference techniques to tie together decisions that should be correlated. In our case, a single *ParentChild* decision might be correlated with another in the same spreadsheet if both the parents and children share the same graphical style. In Figure 1, the *ParentChild* pairs *(26, 27)* and *(32, 33)* should probably be decided in the same way. It may also be possible to correlate *ParentChild* decisions across spreadsheets via type information. For example, we might correlate two *ParentChild* decisions (Male, 18 to 24 years) and (Female, 65 years and over) if we know that Male and Female are both in the "gender" type and that 18 to 24 years and 65 years and over are both "age ranges."

Conditional random fields [26] are a standard approach for joint inference classification tasks, and thus we decide to use a CRF system to model the *automatic extraction* task.

Interactive Repair — The *interactive repair* phase allows the user to check and fix any *ParentChild* decision mistakes made by the system. The *interactive repair* interface presents the initial extraction result from the *automatic extraction*, then allows the user to correct the extracted annotation hierarchies by dragging and releasing nodes into their correct positions (shown in Figure 2).

During *interactive repair*, we assume that a user always fixes extraction errors correctly. For a relatively small number of repairs, this assumption is reasonable. Further, the goal of this paper is to reduce the user's work, not to also address the problem of noisy human-labeled data (although there is crowdsourcing literature on how to ensure trustworthy answers [13].) The maximum number of repairs a user will have to provide to remove all the extraction errors is the number of annotations in the *left* (or *top*) region.

We know that many *ParentChild* decisions are tied. We may be able to squeeze more utility from each user repair by using it to also fix extraction errors the user has not yet explicitly examined and repaired. For example, a user's repair to an extraction error on the *ParentChild* pair (Male, White) should be automatically applied to (Male, Black). Of course, any mechanism that tries to fix extraction errors on the user's behalf could *itself* introduce errors; we call this problem *backtracking* and will discuss it in more detail later.

A ***basic*** method of spreading repairs is to tie the *ParentChild* decisions in one spreadsheet if the parents share the same formatting and so do the children: if a user changes one decision, the system automatically applies the change to the tied ones. But this basic method has some shortcomings: style properties might not hold across spreadsheets, and hierarchies in spreadsheets might use conflicting styles.

Instead, we can reuse the correlations among *ParentChild* decision pairs that we discovered for our *automatic extraction.* While stylistic properties can be used to propagate repair decisions within a single spreadsheet, we can use metadata type information to tie *ParentChild* decisions across multiple spreadsheets in the corpus. Therefore, we again use a CRF (an extended version of the *automatic extractor*

CRF) to incorporate local and correlated information.

In summary, our design decisions lead us to use a general CRF as the technique to support both *automatic extraction* and *interactive repair*.

3. AUTOMATIC EXTRACTION

In this section, we describe how to exploit the different sources of information described above and how to encode the *automatic extraction* as a conditional random field. First, however, we provide a short summary of CRFs.

3.1 Background: Conditional Random Fields

A conditional random field G is a discriminative undirected graphical model [24, 26], primarily used for labeling or segmentation tasks. It describes a joint distribution over a set of n random variables $\mathbf{x} = \{x_1, ..., x_n\}$, where each variable x_i takes a label l_i from a set of labels L. The model captures properties of each variable and dependencies among variables in the graph by defining *potential functions* on cliques of correlated variables. Then, the probability of an assignment of labels to variables can be described as the joint distribution of the potential functions as follows:

$$P(\mathbf{l} \mid \mathbf{x}) = \frac{1}{Z} \prod_{C \in clique(G)} \phi_C(x_C)$$

where Z is a normalization factor to make the product of the clique potentials $\phi_C(x_C)$ into a proper probability.

A common method to define the potentials is as a dot function between the *weight* parameters and a *feature* vector [29]. A node potential captures the features that correspond to a single variable. The node potential is usually defined on a variable x_i as $\theta(x_i) = \mathbf{w_1}^T \mathbf{f}(x_i, l_i)$, where $\mathbf{f}(x_i, l_i)$ is a feature vector and $\mathbf{w_1}$ is the associated weight parameters. Similarly, the edge potential is usually defined on pairwise variables x_i and x_j to describe their correlation as $\theta(x_i, x_j) = \mathbf{w_2}^T \mathbf{f}(x_i, l_i, x_j, l_j)$. Users generally provide domain knowledge via the feature vectors $\mathbf{f}$, while the parameters $\mathbf{w} = \{\mathbf{w_1}, \mathbf{w_2}\}$ are *trained* from labeled data. In the *training* stage, the feature vector is derived from a set of labeled data in order to obtain the optimal value for the *weight* parameters $\mathbf{w}$. In the *inference* stage, the probability $P(\mathbf{l} \mid \mathbf{x})$ can be computed using the trained parameters, and the optimal labeling can be obtained by finding the maximum joint probability.

3.2 Encoding Hierarchy Extraction

We now formally describe our problem and observations. The task of hierarchy extraction is to detect all of the *ParentChild* pairs $P = \{ParentChild(a_i, a_j)\}$ in an annotation region A. One way to model this problem is to create a Boolean variable x to represent a *ParentChild* pair candidate (a_p, a_c) for every annotation pair $a_p, a_c \in A$. Each variable x takes a label $l \in L = \{true, false\}$, and x holds true if a_p is the parent of a_c. For example, Figure 4 shows a portion of the created variables for Figure 1's *left* metadata. Each oval node corresponds to a single boolean *ParentChild* decision. For example, setting the node (18 to 24 years, Male) to true indicates that 18 to 24 years is the hierarchy *parent* of Male.

However, simply enumerating all pairs in a region A can yield thousands of variables. In practice, it is possible to greatly reduce the set of *ParentChild* candidates with a few heuristics. For example, we prioritize *ParentChild* candidates in which the typographic styles of the two nodes differ. We also prioritize pairs that are geometrically close to each other in the spreadsheet. Testing on our 200 testbed spreadsheets for SAUS and WEB, our heuristics permit almost all of the correct *ParentChild* pairs to be encoded as variables in the model, incorrectly filtering out just 0.01% and 0.13% of correct pairs, respectively. Failing to create a node for a true *ParentChild* relationship clearly means that we will predict the child's parent incorrectly. A wrong *ParentChild* prediction is bad, but not catastrophic: the user can still describe the correct relationship during *interactive repair*.

A true *ParentChild* variable may be indicated by the surrounding style and layout information. For example, a variable that describes annotations which are physically close is likelier to be true than a variable that describes annotations that are physically distant. We formulated 32 features from Table 1 for evaluating a *ParentChild* variable, each of which is a binary function. Note that for each *ParentChild* pair candidate, *unary features* apply to the parent and child individually, while *binary features* directly apply to the pair.

3.3 Correlating ParentChild Decisions

ParentChild decisions can be correlated; knowing the assignment of one *ParentChild* variable sheds light on some others. We found the following four types of correlations.

Correlation (i) — Stylistic Affinity. When two *ParentChild* variables in the same spreadsheet have identical visual style for parents and for children, it is likely that the two variables should be decided together. For example in Figure 5 (a), the two *ParentChild* variables ((White, College) and (Male, 18 to 24 years)) should be decided together because the parents (White and Male) share the same typographic style, as do the children (College and 18 to 24 years). We say that two variables have stylistic affinity when the parents and children share a range of visual qualities: alignment, indentation, capitalization, typeface, type size, type style (*i.e.*, bold or italicized), and use of certain special strings (*i.e.*, a colon, a number, or the word "total"). Note that stylistic affinity only makes sense when testing *ParentChild* pairs ***within a single spreadsheet***; different

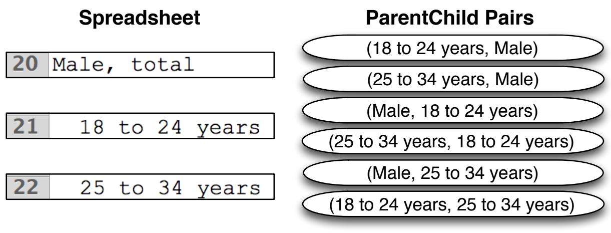


Figure 4: A sample of *ParentChild* variables.

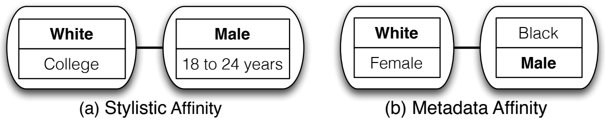


Figure 5: An example of stylistic affinity shown in (a) and metadata affinity shown in (b).

Unary Features	
1	Annotation has underline
2	Annotation contains “total”
3	Annotation contains “:”
4	Annotation is bold
5	Annotation is center aligned
6	Annotation is italic
7	Annotation is numeric
8	Annotation letters are all capitalized
9	Is the first annotation
10	Is the last annotation
Binary Features	
1	Annotation pair is adjacent
2	Annotation pair’s style is adjacent in the region
3	Child’s font size is smaller than parent’s
4	Child’s indentation is greater than parent’s
5	Child’s row index is greater than parent’s
6	Has blank cells in the middle
7	Has middle cell with indentation between the pair’s
8	Has middle cell with indentation larger than the pair’s
9	Has middle cell with indentation less than the pair’s
10	Has middle cell with style different from the pair’s
11	Has middle cell containing “:” or “total”
12	Parent is the root

Table 1: Node potential features.

spreadsheets may have different or contradictory ways of visually indicating the *ParentChild* relationship.

Correlation (ii) — Metadata Affinity. If we have a metadata resource available, we can use it to find additional correlations between *ParentChild* variables both ***within and between*** spreadsheets. For example in Figure 5 (b), the two *ParentChild* candidates, (White, Female) and (Black, Male), should be decided together because the parents (White and Black) belong to the same semantic category *race*; similarly the children (Female and Male) belong to *gender*. The details of how to test whether two variables have metadata affinity are discussed in Section 5.

Correlation (iii) — Adjacent Dependency. If we consider the *ParentChild* pairs of a ***single spreadsheet*** as a sequence, adjacent variables often follow a transition pattern of the labels.

Correlation (iv) — Aggregate Design. There are two further constraints that reflect typical overall spreadsheet design and ensure that the resulting variable assignment yields a legal hierarchy (*i.e.*, a tree).

The first is the *orientation constraint.* Spreadsheets tend to have an “upward” or “downward” orientation; that is, parents do not appear above their children in some cases and below their children in other cases. For example in Figure 4, the pairs (Male, 18 to 24 years) and (25 to 34 years, 18 to 24 years) cannot simultaneously be *true.*

The second is the *one-parent constraint.* We enforce our assumption that *ParentChild* relationships genuinely form a tree; one annotation can only have one parent. Put another way, for all of the variables sharing the same child, only one of them is *true* and the rest are *false.* For example, in Figure 4, (Male, 25 to 34 years) and (18 to 24 years, 25 to 34 years) could not both be true.

3.4 Encoding Automatic Extraction to the CRF

Now we describe how we encode the *ParentChild* pair properties and their correlations into the CRF.

Node Potentials – Each variable x in the CRF represents a *ParentChild* decision, which takes a label $l \in L = \{true, false\}$. The properties that concern just a single *ParentChild* pair discussed in Section 3.2 and Table 1 are easy to encode as *node potentials.* We define the **node potential** $\theta(x, l)$ on each variable x assigned the label l. The node potentials depend on Boolean feature functions $\{f_k(x, l)\}$ that capture specific clues about variable x's compatibility with corresponding label l and trained weights $\{w_k\}$ associated with the feature functions:

$$\theta(x,l) = \sum_k w_k f_k(x,l) \tag{1}$$

Edge Potentials – The correlations (i) (ii) and (iii) mentioned in Section 3.3 are encoded in the CRF as pairwise *edge potentials.* The **edge potential** $\theta(x, l, x', l')$ is defined on two variables x and x' in the CRF on their assignments l and l' if the variables x and x' are found to be correlated in one of the three ways mentioned above. We define,

$$\theta(x,l,x',l') = [\![l = l']\!] \sum_e w_e f_e(x,x') \tag{2}$$

where $[\![l = l']\!]$ takes the value 1 when $l = l'$ and 0 otherwise; $\{w_e\}$ are the associated weights. The edge features $\{f_e(x, x')\}$ test which type of correlation x and x' belong to and whether x and x' have the same child/parent.

Global Potentials – Finally we encode the correlation (iv) mentioned in Section 3.3 as *global potentials.* Let $x = (a_p, a_c)$ and $x' = (a'_p, a'_c)$ be two arbitrary variables in the CRF with the assigned label l and l', and $R(a)$ represents the row number of an annotation a. We now define two **global potentials**: $\phi_a(\mathbf{x}, \mathbf{l})$ to encode the *orientation constraint* and $\phi_b(\mathbf{x}, \mathbf{l})$ to encode the *one-parent constraint*:

$$\begin{aligned}\phi_a(\mathbf{x},\mathbf{l}) = [\![\exists x, x' \in \mathbf{x}\ s.t.\ l = true, l' = true,\\ R(a_p) > R(a_c), R(a'_p) < R(a'_c)]\!]^0_{-\infty}\end{aligned} \tag{3}$$

$$\phi_b(\mathbf{x},\mathbf{l}) = [\![\forall c, \sum_p [\![l = true]\!]^0_1 = 1]\!]^{-\infty}_0 \tag{4}$$

where $[\![C]\!]^{value2}_{value1}$ takes the *value 1* when condition C is true and *value 2* otherwise.

4. INTERACTIVE REPAIR

Recall that the goal of our *interactive repair* is to save user effort by using each explicit user-given repair to fix not just the error in question, but also additional extraction errors that the user never directly inspects. In this section, we describe the *interactive repair* workflow in more detail, plus how to modify the CRF to support the repair process. Finally, we describe the CRF training and inference methods.

4.1 User Repairs

We now discuss our model workflow for *interactive repair.* The system starts by presenting to the user the initial extraction results computed by the *automatic extraction.* The system then enters the *interactive repair* interaction loops (shown in Figure 6). For each loop, the system takes two steps. Step 1 is **review and repair**, in which users explicitly review the current extraction result and repair a single error. Step 2 is **spread repairs**, in which the system applies the user’s input to automatically repair other extraction errors that the user has not yet identified.

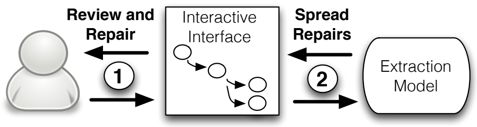


Figure 6: Interaction cycle for *interactive repair*.

```
Algorithm 1 SpreadRepair
Input: All user repairs $R$, and automatic extraction CRF $G$
Output: New assignments $\mathbf{l}$ to all variables of CRF $G$.
1: From user repairs $R$, create repair-induced CRF variables $\mathbf{x_r}$
   with labels $\mathbf{l_r}$ (as described in Section 4.1 Step 1).
2: Build new CRF $G'$ by adding to $G$ the new repair potentials
   based on $\mathbf{x_r}$. $G'$ has the same set of nodes (variables) as $G$.
3: Condition on $\mathbf{x_r}$ and infer assignments $\mathbf{l}$ to $G'$ (and thus, $G$)
```

1. Review and Repair — A user is able to repair an error in the extracted hierarchy by dragging and releasing an annotation node on the interface. One ***user repair*** action changes an annotation's parent from one to another. For example in Figure 2, a user changes the parent annotation of White from Root to Male.

A *user repair* operation has two implications. First, the variable x that represents the new *correct ParentChild* relationship is set to *true*. In the case of Figure 2, the variable (Male, White) is *true*. Second, all the other variables that represent *ParentChild* relationships sharing the same child with x are set to *false*. In the case of Figure 2, variables (Root, White) and (Total smoker, White) are *false*.

As a result, a user's repair to an extraction error yields a set of label assignments to some *ParentChild* variables. Recall that we assume a user always fixes extraction errors correctly, and thus, all the repair induced *ParentChild* labels are assumed to be correct.

2. Spread Repairs — The system now aims to save user effort by repairing other similar extraction errors.

Of course, the system has already given its best extraction estimate in the *automatic extraction* phase, so it does not know where any latent extraction errors are. But we have already used different kinds of affinity to connect two *ParentChild* decisions that are highly likely to share the same label; it is a shame to forget about this information just when the user is providing a new source of correct labels.

It is appealing to spread each user-repaired label on a variable to any other variable that is identified by our affinity correlations (i) and (ii). But simply propagating assignments might introduce errors where none previously exist, which we call the ***backtracking*** problem. We want to leverage the CRF to integrate probability information with the node, edge, and global correlations to prevent the problem.

4.2 Encoding User Repairs with the CRF

Here, we describe how to encode the user repair interaction to the CRF. Algorithm 1 shows the **SpreadRepair** function that is invoked after each user repair operation (described in step 2 of the previous section). First, when a new repair arrives, we translate this new repair and all the previous repairs to the assignments on a set of variables $\mathbf{x_r} = \{x_{r_1}, ..., x_{r_n}\}$ with labels $\mathbf{l_r} = \{l_{r_1}, ..., l_{r_n}\}$. Second, we generate a new CRF G' by adding the *repair potentials* to the original *automatic extraction* CRF G. The repair potentials capture the pairwise correlation between variables, and we describe the repair potentials later. Finally, we condition on the known variables $\mathbf{x_r}$ and infer labels for the variables of G'. The inferred labels are returned as the updated answer.

Note that by adding repair potentials only to nodes that we also condition on, we add information to the inference process without increasing any inference-time computational complexity. The conditioning process essentially removes the observed nodes and their edges prior to the inference [24].

There is nothing in principle that prevents our system from ***backtracking***, unless we can find heuristics to propagate the assignments fully correctly, which is often hard especially on real-world datasets. However, our mechanism is designed to prevent it. First, we only probabilistically propagate known variable assignments to others, via the repair potentials. Second, this probabilistic repair information is combined with all our previous information sources: the node potentials, edge potentials and global potentials. The hope is that adding high quality new information to the *automatic extraction* CRF (instead of treating spreading repairs as a non-probabilistic post-processing stage) will yield better outcomes overall.

We now discuss how to generate the repair potentials.

Repair Potentials — The **repair potential** $\varphi(x, l, x_r, l_r)$ describes the likelihood that the repaired node's label should be spread to a similar *ParentChild* node. A repair potential exists between an observed variable $x_r \in \mathbf{x_r}$ and a variable $x \in \mathbf{x}$ if x_r and x exhibit either stylistic affinity or metadata affinity. In other words, the repair potentials do not introduce any novel edges to the CRF: the edges of repair potentials are a subset of the edges derived from correlations (i) and (ii). The repair potentials are defined as follows:

$$\begin{aligned}\varphi(x, l, x_r, l_r) &= [\![Stylistic(x, x_r)]\!] f_s(x, l, x_r, l_r) \\ &+ [\![Metadata(x, x_r)]\!] f_m(x, l, x_r, l_r)\end{aligned} \tag{5}$$

$[\![C]\!]$ takes the value 1 when condition C is true; otherwise 0. $Stylistic(x, x_r)$ and $Metadata(x, x_r)$ test whether x and x_r have stylistic or metadata affinity. The two feature functions f_s and f_m weigh the strength of influence from observed variables to unobserved ones. They characterize how similar the unobserved variables are to the observed ones. To be precise, we define $f_s(x, l, x_r, l_r) = log P_s(x = l \mid x_r = l_r)$. where $P_s(x = l \mid x_r = l_r)$ represents the probability of a variable x taking the label l once we observe a variable x_r with the label l_r. This probability can be derived from training data. For example, in the training data, among 1000 *stylistic affinity* edges detected, 900 of them connect two variables with the same assignment. We then set $P_s(x = true|x_r = true) = 0.9$ and $P_s(x = false|x_r = true) = 0.1$. The f_m potentials are defined in the same way.

Summary — We can now formally define the spreadsheet annotation hierarchy extraction CRF framework, which supports both *automatic extraction* and *interactive repair*.

DEFINITION 2. *(Annotation Hierarchy Extraction CRF) Let G be a CRF that has a set of variables $\mathbf{x} = \{x_1, ..., x_n\}$ where each $x_i \in \mathbf{x}$ represents a* ParentChild *candidate in an* annotation region *and takes a label l_i from L =*{true, false}*. If l_i =* true *then the* ParentChild *candidate represented by x_i is true; otherwise it is false. We define node potentials (Equation 1), edge potentials (Equation 2), global potentials (Equation 3 and 4), and repair potentials (Equation 5) in the CRF. The joint probability distribution of the CRF is:*

$$P(\mathbf{l} \mid \mathbf{l_r}, \mathbf{x}) = \frac{1}{Z(\mathbf{w})} exp(\sum_x \theta(x,l) + \sum_x \sum_{x'} \theta(x,l,x',l') + \sum_{k \in \{a,b\}} \phi_k(\mathbf{x},\mathbf{l}) + \sum_x \sum_{x_r} \varphi(x,l,x_r,l_r))$$

4.3 Training and Inference

In this section, we discuss how to train model parameters and infer assignments to variables in the CRF.

4.3.1 Parameter Estimation

According to Definition 2, we only have unknown parameters for *node* and *edge potentials*. Assuming that no user repairs are involved, we can write the joint probability as,

$$\frac{1}{Z(\mathbf{w})} exp(\sum_x \theta(x,l) + \sum_x \sum_{x'} \theta(x,l,x',l') + \sum_{k \in \{a,b\}} \phi_k(\mathbf{x},\mathbf{l}))$$

Let $\mathbf{w} = \{w\}$ be the set of parameters for *node* and *edge potentials*. Given training data $D = \{\mathbf{x}, \mathbf{l}\}$ that describes hand-labeled correct hierarchies of the training spreadsheets, we estimate $\mathbf{w}$ for node and edge potential functions, $\theta(x,l)$ and $\theta(x,l,x',l')$. A common choice of regularization to avoid overfitting is to add a penalty on weight vectors, based on the Euclidean norm of $\mathbf{w}$ and on a *regularization parameter* $\frac{1}{2\sigma^2}$. The goal is to maximize the regularized log likelihood:

$$\max_{\mathbf{w}} \sum_x \theta(x,l) + \sum_x \sum_{x'} \theta(x,l,x',l') - logZ(\mathbf{w}) - \sum_i \frac{w_i^2}{2\sigma^2} + C$$

where C is a constant. This is a standard form for CRF parameter estimation, and known techniques, such as conjugate gradient and L-BFGS, can be used to find the optimal parameters for this formula. Previous work [24, 26] discusses this optimization problem and its solution in more detail.

4.3.2 Inference Technique

The general graph CRF described in Definition 2 poses a serious computational challenge. Inference is NP-hard if no assumptions are made about the structure of the CRF [11], yet our application requires that we infer labels after each user repair to redisplay the updated hierarchy. In order to infer variables in interactive time, we first simplify the CRF.

Model Simplification — The potential stumbling blocks to efficient inference are the edge and global potentials. (The repair potentials do not complicate the inference because the conditioning algorithm [24] erases observed variables along with all the repair potential edges.) The edge potentials alone can yield more than a million edges on a graph with 37,386 nodes derived from just 100 randomly-chosen WEB spreadsheets (see Table 4 for details).

We considered two methods for conducting inference in a limited amount of time: running the tree-reweighted belief propagation algorithm [25] on the full CRF graph, or running an exact inference method on a simplified tree-structured CRF. Our experiments show that when running on a CRF derived from 100 random SAUS spreadsheets and repeating this process 10 times, tree-reweighted belief propagation is 48 times slower and 5.4% worse on F1 than the tree-structured CRF. Thus, at inference time we convert our CRF graph into a tree-structured CRF (which has also been demonstrated to significantly outperform the existing linear-chain CRF model [32]).

Algorithm 2 EnforcedTreeInference

Input: The variables $\mathbf{x} = \{x\}$ and the annotations $A = \{a_1, ..., a_N\}$ in an annotation region.
Output: The *ParentChild* pairs $P = \{(a_p, a_c)\}$ in the annotation hierarchy and its confidence $confidence$.

```
 1: $P \leftarrow \{\}$, $confidence \leftarrow 0$
 2: for each $a_c \in A$ do
 3:     $maxprob \leftarrow 0$, $a_{p_0} \leftarrow root$
 4:     for each $a_p \in A$ do
 5:         Find $x \in \mathbf{x}$ for the ParentChild pair $(a_p, a_c)$
 6:         Obtain the probability $cprob$ that $x = true$
 7:         if $cprob > maxprob$ then
 8:             $maxprob \leftarrow cprob$, $a_{p_0} \leftarrow a_p$
 9:         end if
10:     end for
11:     $P \leftarrow P \cup \{(a_{p_0}, a_c)\}$
12:     $confidence \leftarrow confidence + log(maxprob)$
13: end for
```

It is not easy to find the tree-structured CRF that yields the highest-quality results. Exhaustively enumerating all the possible trees in a graph with more than a million edges and 37,000 nodes is impractical. We simply randomly sample edges from each type of pairwise correlation (stylistic, metadata, and adjcency), rejecting any edge that would induce a cycle. We terminate when all nodes are connected. We add all possible metadata edges before adding any stylistic ones, and add all stylistic ones before adding any adjacency ones. We found experimentally that this ordering helped slightly, though different orderings do not change F1 very much: testing on 100 random spreadsheets of SAUS, different orderings changed F1 from 0.8808 to 0.8867 and from 0.8237 to 0.8363 when testing on WEB.

Inference — We can now present our method for approximating the CRF's optimal assignment. First, we build the CRF with *node* potentials, tree-structured *edge potentials*, and all the *repair potentials* if there exist any. Given a set of observed variables $\mathbf{x_r}$ with labels $\mathbf{l_r}$ translated from users' repairs (we assume $\mathbf{x_r}$ is empty if no repairs are observed), the *conditioning* algorithm yields a forest-structured CRF.

Second, we run a standard inference algorithm on this new CRF to obtain the assignment to all the variables. Because the CRF is now a forest-structured CRF, a variety of existing algorithms, such as belief propagation, can perform exact inference on such a structure.

Finally, we treat the *global potentials* as a post-processing stage to ensure that the inferred variable assignment yields legal hierarchical trees for the input annotation regions. The goal of *global potentials* is to handle the *orientation* and *one-parent* constraints. Thus, we first enumerate all of the *ParentChild* candidates of each orientation, "upward" or "downward," and compute two separate annotation hierarchies with **EnforcedTreeInference**, seen in Algorithm 2. For all the *ParentChild* candidates with a given annotation as the child, the algorithm selects the one with the maximal probability (derived from the CRF), thereby handling the *one-parent constraint*. We obtain two possible hierarchies, one "upward" and one "downward," each with computed $confidence$. We select the one with the higher confidence to handle the *orientation constraint*. Therefore, our algorithm yields a legal hierarchical tree for an annotation region.

5. THE METADATA RESOURCE

A critical part of both *automatic extraction* and *interactive repair* is detecting metadata affinity. As described in Section 3.3, *ParentChild* variables might be correlated because they describe data belonging to one semantic category. (Recall that we tie the *ParentChild* decisions for (White, Female) and (Black, Male) if we know the two parents fall into the same category, and so do the two children.) This information is useful for examining annotations within a single spreadsheet, and is the *only* way to tie *ParentChild* decisions across multiple spreadsheets.

General-purpose schema resources, such as Google's Freebase [4], can be used to detect metadata affinity between two annotations. But spreadsheet domains can be quite narrow, possibly specific to the institution or even the individual who originated the spreadsheet. Consider a payroll spreadsheet that assumes the reader knows which employees belong to which departments, or a spreadsheet about industrial production that assumes the reader knows which chemicals arise from which processes. This kind of hyper-specific metadata information is unlikely to exist in general-purpose resources.

Fortunately, we are able to synthesize a *domain-specific metadata resource* from a corpus of spreadsheets. Our central observation is that any useful category of annotations — whether a general-purpose one like gender or a hyper-specific one such as chemicalPrecursor — will likely appear in many datasets. Further, annotations drawn from the same category (such as Male and Female) often appear as siblings in an extracted annotation hierarchy. We measure whether two annotations belong to the same category by testing how strongly the annotations appear as siblings in a large number of extracted hierarchies. We perform the test as follows:

1. Extract all annotation hierarchies from a corpus of spreadsheets using a simple classifier or a version of our *automatic extractor* that does not use metadata information. For each parent annotation, we create a *sibling set* that contains all of its child annotations.
2. Count the number of sibling sets where an annotation a is observed. Divide by the number of sibling sets to obtain $p(a)$, the probability that a randomly chosen sibling set contains a.
3. Count the number of sibling sets where the annotation pair a_i and a_j co-occur together. Divide by the number of sibling sets to obtain $p(a_i, a_j)$, the probability that a randomly chosen sibling set contains both a_i and a_j.

We can then measure the extent to which two annotations a_i and a_j are observed as siblings (and thus are likely to be in the same category) by computing the pointwise mutual information (PMI): $PMI(a_i, a_j) = log\frac{p(a_i,a_j)}{p(a_i)p(a_j)}$. We now formally define *metadata affinity* based on the PMI score.

Definition 3. *(Metadata Affinity) Let $x_1 = (a_{p1}, a_{c1})$ and $x_2 = (a_{p2}, a_{c2})$ be two variables in the CRF. The two variables x_1 and x_2 have* **metadata affinity** *if and only if $PMI(a_{p1}, a_{p2}) > \delta$ and $PMI(a_{c1}, a_{c2}) > \delta$, where δ is a predefined threshold.*

6. EXPERIMENTS

We now evaluate the performance of *automatic extraction* (Section 3) and *interactive repair* (Section 4). We also evaluate the quality of our *metadata resource* (Section 5).

6.1 Experimental Setup

Our experiments are based on two spreadsheet corpora:

- ***SAUS*** – The 2010 Statistical Abstract of the United States (SAUS) consists of 1,369 spreadsheet files totaling 70MB. We downloaded the dataset from the U.S. Census Bureau. It covers a variety of topics of general public interest, such as state-level finances, educational attainment, levels of public health, and so on. The data come from different sources inside the government, but to the human eye appears uniformly high in quality of design and content.

- ***WEB*** – Our Web dataset consists of 410,554 Microsoft Excel files from 51,252 distinct Internet domains. They total 101 GB. We found the spreadsheets by looking for Excel-style file endings among the roughly 10 billion URLs in the ClueWeb09 Web crawl [10]. The data come from many different sources and to a human appears to have a wide range in quality.

From each of the two datasets, SAUS and WEB, we randomly selected 200 *hierarchical spreadsheets*. We call these test sets SAUS **R200** and WEB **R200**. We constructed them by randomly sampling from SAUS or WEB and retaining only the hierarchical ones (*i.e.*, ones that have either hierarchical *left* or *top* annotations). In addition, we constructed a series of topic-specific test sets. For SAUS, we used government-provided category labels to identify spreadsheets for each of three topic areas: **health**, **finance**, and **transportation**; we chose 10 random hierarchical spreadsheets from each topic. For WEB, we used URL domain names as a rough proxy for the category label, choosing 10 random hierarchical spreadsheets from each of **bts.gov**, **usda.gov**, and **nsf.gov**. We asked a human expert to manually examine the above spreadsheets and create ground truth hierarchies. Details about the test sets are shown in Table 2.

Our system uses a mix of code from several languages and projects. We used the Python xlrd library to access data and formatting details of each spreadsheet file. Our CRF model was implemented with UGM [35]. We used the Weka [18] packages to build logistic regression and other classifiers. We obtained WEB spreadsheets using the Nutch crawler.

6.2 Automatic Extraction

In this section, we evaluate the performance of the *automatic extraction* phase. We evaluate the *automatic extraction*'s accuracy in predicting correct *ParentChild* relationships by using standard metrics of Precision, Recall, and F1. We trained and tested *automatic extraction* using SAUS **R200** and WEB **R200**. We randomly split each of the two datasets equally for training and testing. We trained the parameters on the training set and constructed one CRF for the test set. We repeated this split-and-test process 10 times, computing average Precision, Recall and F1.

Automatic Models — We compared the basic method mentioned in Section 2.2 (which we call AutoBasic) with four different configurations of the *automatic extraction* CRF. Recall that AutoBasic only uses alignment and indentation information to infer *ParentChild* pairs. Because we are not aware of any previous system that aims to solve the hierarchy extraction task, AutoBasic serves as a commonsense engineer-driven baseline for comparison.

Dataset		Hierarchy Levels			# Left Annotations		
		Min	Mean	Max	Min	Mean	Max
SAUS	R200	2	3.8	8	4	37.8	224
	health	2	3.6	6	12	34.5	76
	fin.	3	3.7	6	6	32.4	81
	trans.	3	4.0	8	5	36.1	73
WEB	R200	2	3.4	10	2	59.3	669
	bts	2	2.6	4	4	10.7	26
	nsf	2	4.0	7	9	83.9	331
	usda	2	3.2	4	5	34.5	56

Table 2: Basic statistics of our eight test sets.

Dataset	Methods	Precision	Recall	F1
SAUS	AutoBasic	0.4641	0.4641	0.4641
	AutoLR	0.8753	0.8750	0.8751
	AutoEdge	0.8801	0.8787	0.8794
	AutoGlobal	0.8834	0.8834	0.8834
	AutoFull	**0.8860**	**0.8860**	**0.8860**
WEB	AutoBasic	0.4736	0.4736	0.4736
	AutoLR	0.7886	0.7898	0.7892
	AutoEdge	0.7979	0.7968	0.7973
	AutoGlobal	0.8122	0.8122	0.8122
	AutoFull	**0.8327**	**0.8327**	**0.8327**

Table 3: Performance of the *automatic extractor* on SAUS **and** WEB **R200 datasets.**

The following four versions of the CRF demonstrate the power of each component of our *automatic extractor*. AutoLR uses node potentials only (with no edge or global potentials, the model is equivalent to the logistic regression, or LR, method)[4]. AutoEdge uses node potentials and edge potentials. AutoGlobal uses node potentials and global potentials. Finally, AutoFull uses all three potential types and reflects the entire contents of Section 3.4. These four methods all use the extraction features mentioned in Table 1.

For AutoLR and AutoEdge, it is possible to threshold the probability to trade off between precision and recall. We varied the threshold from 0 to 1 by 0.02 and chose a threshold to maximize F1. For the rest three methods, there is no such flexibility, as the algorithm always selects the parent with the maximum *ParentChild* probability for each child.

Table 3 shows the performance of the five methods. We can see that all of our four CRF models significantly outperformed the baseline AutoBasic. Both partial models — AutoEdge and AutoGlobal — performed better than AutoLR, indicating that both *edge* and *global potentials* independently helped to improve the performance of *automatic extraction*. AutoFull, the model that includes all three potential types, is the best of all (though AutoFull's margin is small in the case of SAUS). We noticed that many extraction errors are due to contradictory spreadsheet formatting; designers of different spreadsheets may have conflicting designs, but even the format within one spreadsheet may not be consistent.

Training Data — Labeled *ParentChild* pairs are relatively expensive to obtain and are, thus, rare. We wanted to know if our supply of training data was limiting the *automatic extractor*'s accuracy. We conducted a test in which we artificially constrained the training set size derived from SAUS **R200** and WEB **R200**, building a series of *automatic extraction* CRFs with varying amounts of training data. Figure 7 shows the F1 of the *ParentChild* pairs for AutoFull as we change the size of the training set. The growth in both SAUS and WEB accuracy plateaus after a certain size. This

[4]We also tried support vector machines and other non-joint-inference techniques, but they offered no significant gains over AutoLR.

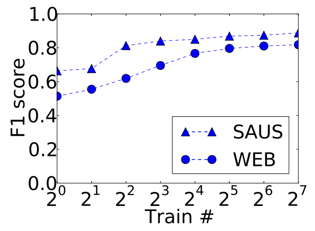


Figure 7: Performance for *automatic extractor* using different amounts of training data.

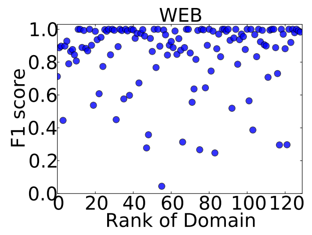


Figure 8: Performance for *automatic extractor* on different domains in WEB**.**

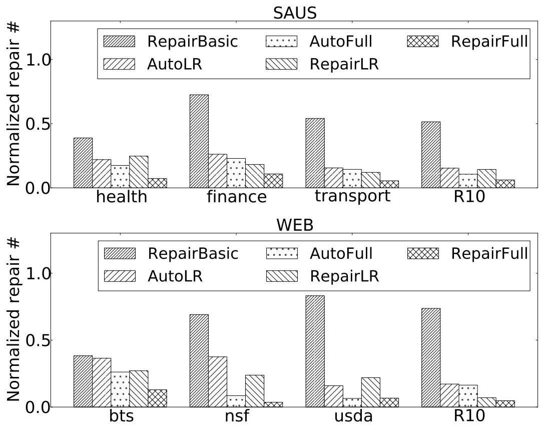


Figure 9: The normalized repair number for *interactive repair* on SAUS **and** WEB **test sets.**

analysis does not mean more training data cannot help, but does indicates that additional gains will likely be expensive.

Domain Sensitivity — We also examined whether the WEB *automatic extractor*'s accuracy varies with the quality of the spreadsheet. It is difficult to precisely describe a spreadsheet's quality, so as a proxy we use the rank of the spreadsheet URL's Internet domain, when sorted in descending order of the number of spreadsheets hosted by the domain. Figure 8 shows the average F1 within each Internet domain's spreadsheets. We followed the same training and testing procedure as in the ***Automatic Models*** part above. The figure shows that the publisher's rank (or the quantity of spreadsheets it publishes) does not correlate with extraction performance. However we *did* find that spreadsheets from lower ranked domains are less likely to pass our initial "hierarchical data frame spreadsheet" filter.

In summary, our system shows substantially better performance than the baseline AutoBasic method, a 91% improvement in F1 on SAUS and a 76% improvement in F1 on WEB. If our user fixes any extraction error by hand, our system amounts to a large reduction in the amount of work the user must perform. We now turn to *interactive repair* to shrink the user's burden even further.

6.3 User Repairs

We now evaluate the performance of the *interactive repair* phase. We use the eight datasets described in Section 6.1. For each **R200** of SAUS and WEB, we again randomly split the dataset into 100 training spreadsheets and

		Sheet #	Node #	Correlation Edge # (×1000)		
				Stylistic	Metadata	Total
SAUS	train	100	11269	87.5	115.7	177.6
	health	10	874	4.9	1.7	5
	fin.	10	1228	8.6	5.5	11.1
	trans.	10	1334	9.5	5.7	12.3
	R10	100	13866	144.4	43.3	161.3
WEB	train	100	31925	724.2	566.9	1069.0
	bts	10	249	0.5	0.0	0.5
	nsf	10	10698	265.1	22.9	283.3
	usda	10	1786	15.1	1.7	15.1
	R10	100	37386	1522.0	289.8	1677.6

Table 4: Basic statistics for each test set's *interactive repair* CRF.

100 testing spreadsheets. We further randomly split the 100 testing spreadsheets into 10 subgroups with 10 spreadsheets in each, as **R10**; we then averaged the performance results over the 10 subgroups. We created one CRF for each test set (health, finance, *etc*), except **R10**, where we created one CRF for each subgroup. Table 4 shows basic statistics for the *interactive repair* CRFs constructed for our test sets.

The metric of success for *interactive repair* is the amount of user work reduced when compared to simply fixing all the errors made by *automatic extractor*. We evaluate the amount of user effort by counting the required number of drag-and-drop repair operations to fix all the extraction errors in an annotation hierarchy, via our visual repair tool (seen in Figure 2). In the experiments, we simulated a user who randomly chooses extraction errors to repair, and who never makes a mistake. The user repairs errors until no errors remain. For each dataset, we ran this process 20 times and counted the average number of repairs performed. Notice that the maximum number of possible repair operations for a given hierarchy is the number of annotations in it.

For each result shown in Figure 9, Figures 10 and 11, we normalize the number of required repairs by the maximum possible number of repairs in that dataset (*i.e.*, the number of annotations). Thus, smaller bars are better, and results should be comparable across datasets.

Repair Models — We evaluated six different versions of our extraction system. AutoLR and AutoFull are the entirely automatic extractors as described in the above section; we assume a user simply fixes all of their extraction errors one after another. RepairLR, RepairEdge, RepairGlobal and RepairFull are created by adding repair potentials to the previous four *automatic extraction* models. RepairFull is the full system described in Section 4. We compared these techniques against a baseline method RepairBasic mentioned in Section 2.2, which simply spreads the repair-induced *ParentChild* label on a variable to another when the two variables share the same typographic style in one spreadsheet.

Figure 9 shows the normalized number of repair operations of different *interactive repair* systems. RepairFull performed the best of all, requiring just *7.2%* of the maximum number of possible repairs when averaged over all test sets. In contrast, AutoFull (itself a dramatic improvement over the *automatic extraction* baseline) requires *15.4%* of the maximum; our exploitation of user repairs thus allows us to reduce the user burden by an additional 53%. AutoLR, an automatic extractor without joint inference, yields an even worse average of *23.3%*; we improve by 69%.

Note that applying user repair information naively yields terrible results: RepairBasic requires *60.2%* of the maximum possible number of repairs, much worse than even AutoLR.

In all the datasets, RepairFull always improves or matches AutoFull, which indicates that our repair mechanism is genuinely beneficial to users; we managed to prevent backtracking and did not create more work for users. The same is *not* true for AutoLR *vs* RepairLR, which backtracks in the cases of SAUS/health and WEB/usda.

We further investigated *interactive repair* by considering different possible configurations of the *interactive repair* CRF on different test sets (shown in Figure 10). The Figure shows that both edge and global potentials are useful in reducing user burden, and using all of them helps the most.

Spreadsheet Grouping — We also investigated the influence of two spreadsheet grouping methods on *interactive repair* performance. (1) ***By topic***: We group spreadsheets according to their human-given topic labels (such as finance and health) or their URL hostnames (such as bts.gov and nsf.gov); and (2) ***By Jaccard similarity***: We compute the clusters by creating a graph in which each spreadsheet is a node, and edges exist when two spreadsheets have Jaccard similarity (computed over the non-numeric strings from each spreadsheet) greater than a threshold of 0.6. We find all weakly connected components in the graph as the spreadsheet groups. Note that grouping spreadsheets should only impact metadata affinity, as metadata affinity is the only way to connect *ParentChild* decisions across spreadsheets.

For both SAUS and WEB, we ran each grouping technique, then randomly selected 3 groups of size 2, 3 groups of size 5, and 3 groups of size 10. For each group, we first built one RepairFull CRF on this group of spreadsheets and computed the number of repairs required to eliminate all the extraction errors. We then compared against the sum of repairs needed by RepairFull when running on each spreadsheet of the group in isolation. We found that grouping by topic only reduces repairs up to 5.8% on SAUS and 2.1% on WEB, while grouping by Jaccard similarity reduces repairs by up to 64.0% in SAUS and 84.6% in WEB.

Thus, Jaccard similarity grouping yields a massive reduction in necessary user repairs when compared to topic grouping. We did not present these results in Figures 9 and 10 because we believe that highly coherent clusterings will only be possible in certain situations. Shared spreadsheet templates is one such situation; another is when the metadata resource is of especially high quality (perhaps even curated by hand), allowing *interactive repair* to find otherwise invisible connections among independent spreadsheets.

Metadata Resources – The quality of our metadata resource clearly impacts metadata affinity. Figure 11 shows the normalized number of repairs required by different metadata resource configurations of RepairFull, when run on Jaccard-clustered spreadsheets mentioned above in ***Spreadsheet Grouping***. We compared the approach based on our metadata resource from Section 5 (RepairFull-Metadata) against a no-metadata technique (RepairFull-Style) and a technique that uses Freebase to discover metadata affinity (RepairFull-Freebase). (In that last case, two annotations have metadata affinity if they share the same Freebase topic.) The figure shows that in all cases, our RepairFull-Metadata technique performs the best, usually followed by RepairFull-Freebase. On average, our induced metadata resource reduces user effort by 34.4% when compared to the Freebase resource. Note that some of the spreadsheets we process are on extremely technical

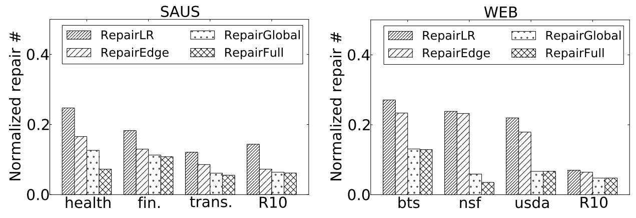


Figure 10: The normalized repair number for four *interactive repair* configurations.

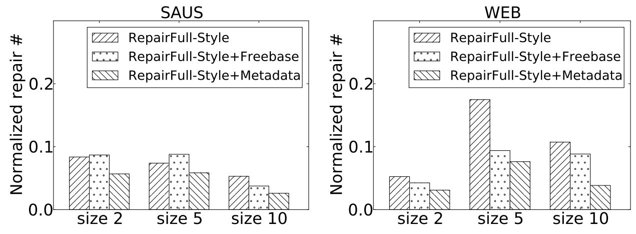


Figure 11: The normalized repair number required by different configurations of metadata links.

topics (such as currency trading, health care, and minerals processing) that are unlikely to be captured in a general-purpose metadata resource such as Freebase.

Runtime Performance — After each user repair operation, users have to wait for the model to re-compute the new result. In our experiments, each repair's inference took 0.7s on **R10** in SAUS and 3.2s on **R10** in WEB on average. All other test datasets, except for **nsf**, took less than 0.7s, whereas **nsf** took 4.7s. These results indicate that interactive repair is computationally feasible, at least for relatively small spreadsheet sets.

Overall, we have demonstrated that our RepairFull extraction system can extract accurate spreadsheet hierarchies using just 7.2% of the maximum possible human effort, a reduction of 53% compared to AutoFull, our automatic extraction system (itself an significant improvement over previous *automatic extraction* techniques). These numbers apply to real-world datasets; in certain cases where spreadsheets share a large amount of metadata, we can improve the factor even further. Moreover, our system works well on domain-specific datasets with no explicit user-provided metadata.

7. RELATED WORK

There are three main areas of related work: spreadsheet management, tabular data extraction, and systems for programming by demonstration.

Spreadsheet Management – Existing approaches for transforming spreadsheet data into databases fall into a few broad categories. First, *rule-based* approaches [2, 17, 20, 27] often require users to learn a newly defined language to describe the transformation process. The approaches are flexible but often require explicit conversion rules that are difficult and time-consuming for the user to compose. Second, there is a range of *visualization* systems [30, 31] that help the user navigate and understand spreadsheets with visualization techniques, but the mechanisms are not able to extract relational data from spreadsheets. Finally, *automated* approaches are the most similar to ours. Abraham and Erwig [1] attempted to recover spreadsheet tuples, but they focused on a simple type of spreadsheets and did not address the hierarchical structures that are key to understanding a huge portion of the online spreadsheet data. Cunha *et al.* [12] assumed some straightforward spreadsheet layouts and primarily focused on the problem of data normalization.

Recent research on spreadsheets also attempted to build *database-like operators* on a *spreadsheet-style interface* [28, 34], but these systems assume the underlying data is already in a relational format and that it cannot be directly used to manage the large number of spreadsheets in the Web.

Tabular Data Extraction – There has been a large amount of work centered on extracting tabular data on the Web [5, 6, 36, 37]. Most of these projects have focused on the details of identifying data-centric tables or on applications that can be built on top of them. HTML tables likely contain hierarchical-style data examples, but we are not aware of any research to date focused on this problem. A few recent projects have aimed to process some domain-specific metadata [36, 37] through analysis of multiple HTML tables; their compiled metadata is similar to the metadata resource we compile. However, our compiled metadata is more specific for spreadsheet data and is able to improve our extractor's performance.

Programming By Demonstration – The *interactive repair* component of our work is part of an intellectual thread that ties programming by demonstration [16, 22, 23, 33], mixed-initiative systems [19], and incorporation of user feedback into extraction systems [7]. Many of these systems are driven by a programming language that the user must learn; our system does not require the user to learn a language, just to use a "drag-and-release" interface. Moreover, we are able to sidestep a full language by using the initially extracted domain-specific metadata to both guide user feedback and find links between repair opportunities.

8. CONCLUSIONS AND FUTURE WORK

We have described a *semiautomatic* framework for extracting data from spreadsheets. This system can derive accurate extractions with dramatically lower user effort than required by a traditional system. It should enable individuals and organizations to better exploit the large amount of data currently locked away in spreadsheet files.

There are several promising avenues for future work. In the short term, we would like to apply our extraction system to other kinds of grid-oriented data, such as Web HTML tables, text-embedded reports, and datasets found in academic papers. More ambitiously, we would like to automatically and preemptively integrate spreadsheets with all the data resources in an organization: relational databases, unstructured documents, even data-centric images, such as plots. The resulting cross-type integrated database could be used as the basis of a general query tool that can ignore distracting details of how each data item happened to be stored.

9. REFERENCES

[1] R. Abraham and M. Erwig. Ucheck: A spreadsheet type checker for end users. *J. Vis. Lang. Comput.*, 18(1):71–95, 2007.

[2] Y. Ahmad, T. Antoniu, S. Goldwater, and S. Krishnamurthi. A type system for statically detecting spreadsheet errors. In *ASE*, pages 174–183, 2003.

[3] P. Blattner and L. Stewart. Microsoft excel 2000 functions in practice. *QUE*, 1999.

[4] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD*, pages 1247–1250, 2008.

[5] M. J. Cafarella, A. Halevy, D. Z. Wang, E. Wu, and Y. Zhang. Webtables: Exploring the power of tables on the web. In *VLDB*, pages 538–549, 2008.

[6] M. J. Cafarella, A. Y. Halevy, and N. Khoussainova. Data integration for the relational web. *PVLDB*, 2(1):1090–1101, 2009.

[7] X. Chai, B.-Q. Vuong, A. Doan, and J. F. Naughton. Efficiently incorporating user feedback into information extraction and integration programs. In *SIGMOD*, pages 87–100, 2009.

[8] Z. Chen and M. Cafarella. Automatic web spreadsheet data extraction. In *SSW*, Trento, Italy, 2013. ACM.

[9] Z. Chen, M. Cafarella, J. Chen, D. Prevo, and J. Zhuang. Senbazuru: A prototype spreadsheet database management system. In *VLDB Demo*, 2013.

[10] 2009. ClueWeb09, http://lemurproject.org/clueweb09.php/.

[11] G. F. Cooper. The computational complexity of probabilistic inference using bayesian belief networks. *Artif. Intell.*, 42(2-3):393–405, 1990.

[12] J. Cunha, J. Saraiva, and J. Visser. From spreadsheets to relational databases and back. In *PEPM*, pages 179–188, 2009.

[13] O. Dekel and O. Shamir. Vox populi: Collecting high-quality labels from a crowd. In *COLT*, 2009.

[14] H. Elmeleegy, J. Madhavan, and A. Y. Halevy. Harvesting relational tables from lists on the web. *VLDB J.*, 20(2):209–226, 2011.

[15] W. Gatterbauer, P. Bohunsky, M. Herzog, B. Krüpl, and B. Pollak. Towards domain-independent information extraction from web tables. In *WWW*, pages 71–80, 2007.

[16] P. J. Guo, S. Kandel, J. M. Hellerstein, and J. Heer. Proactive wrangling: Mixed-initiative end-user programming of data transformation scripts. In *UIST*, pages 65–74, 2011.

[17] M. Gyssens, L. V. S. Lakshmanan, and I. N. Subramanian. Tables as a paradigm for querying and restructuring. In *PODS*, pages 93–103, 1996.

[18] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *SIGKDD Explorations*, 11(1):10–18, 2009.

[19] E. Horvitz. Principles of mixed-initiative user interfaces. In *CHI*, pages 159–166, 1999.

[20] V. Hung, B. Benatallah, and R. Saint-Paul. Spreadsheet-based complex data transformation. In *CIKM*, pages 1749–1754, 2011.

[21] D. F. Huynh, R. C. Miller, and D. R. Karger. Potluck: Data mash-up tool for casual users. In *ISWC/ASWC*, pages 239–252, 2007.

[22] Z. G. Ives, C. A. Knoblock, S. Minton, M. Jacob, P. P. Talukdar, R. Tuchinda, J. L. Ambite, M. Muslea, and C. Gazen. Interactive data integration through smart copy & paste. In *CIDR*, 2009.

[23] S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer. Wrangler: interactive visual specification of data transformation scripts. In *CHI*, pages 3363–3372, 2011.

[24] D. Koller and N. Friedman. *Probabilistic Graphical Models - Principles and Techniques.* MIT Press, 2009.

[25] V. Kolmogorov. Convergent tree-reweighted message passing for energy minimization. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(10):1568–1583, 2006.

[26] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*, pages 282–289, 2001.

[27] L. V. S. Lakshmanan, S. N. Subramanian, N. Goyal, and R. Krishnamurthy. On query spreadsheets. In *ICDE*, pages 134–141, 1998.

[28] B. Liu and H. Jagadish. A spreadsheet algebra for a direct data manipulation query interface. In *ICDE*, pages 417–428, 2009.

[29] R. Pimplikar and S. Sarawagi. Answering table queries on the web using column keywords. *PVLDB*, 5(10):908–919, 2012.

[30] R. Rao and S. K. Card. The table lens: merging graphical and symbolic representations in an interactive focus + context visualization for tabular information. In *CHI*, pages 318–322, 1994.

[31] M. Spenke, C. Beilken, and T. Berlage. Focus: The interactive table for product comparison and selection. In *UIST*, pages 41–50, 1996.

[32] J. Tang, M. Hong, J.-Z. Li, and B. Liang. Tree-structured conditional random fields for semantic annotation. In *ISWC*, pages 640–653, 2006.

[33] R. Tuchinda, P. A. Szekely, and C. A. Knoblock. Building data integration queries by demonstration. In *IUI*, pages 170–179, 2007.

[34] J. Tyszkiewicz. Spreadsheet as a relational database engine. In *SIGMOD*, pages 195–206, 2010.

[35] 2007.
http://www.di.ens.fr/ mschmidt/Software/UGM.html.

[36] M. Yakout, K. Ganjam, K. Chakrabarti, and S. Chaudhuri. Infogather: entity augmentation and attribute discovery by holistic matching with web tables. In *SIGMOD*, pages 97–108, 2012.

[37] M. Zhang and K. Chakrabarti. Infogather+: semantic matching and annotation of numeric and time-varying attributes in web tables. In *SIGMOD*, pages 145–156, 2013.