Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence

Synthesizing Union Tables from the Web

Xiao Ling*
xiaoling@cs.washington.edu
University of Washington

Alon Halevy and **Fei Wu** and **Cong Yu**
{halevy,wufei,congyu}@google.com
Google Research

Abstract

Several recent works have focused on harvesting HTML tables from the Web and recovering their semantics [Cafarella *et al.*, 2008a; Elmeleegy *et al.*, 2009; Limaye *et al.*, 2010; Venetis *et al.*, 2011]. As a result, hundreds of millions of high quality structured data tables can now be explored by the users. In this paper, we argue that those efforts only scratch the surface of the true value of structured data on the Web, and study the challenging problem of *synthesizing tables from the Web*, i.e., producing *never-before-seen* tables from raw tables on the Web. Table synthesis offers an important semantic advantage: when a set of related tables are combined into a single union table, powerful mechanisms, such as temporal or geographical comparison and visualization, can be employed to understand and mine the underlying data holistically.

We focus on one fundamental task of table synthesis, namely, *table stitching*. Within a given site, many tables with *identical* schemas can be scattered across many pages. The task of table stitching involves combining such tables into a single meaningful *union* table and identifying extra attributes and values for its rows so that rows from different original tables can be distinguished. Specifically, we first define the notion of *stitchable tables* and identify collections of tables that can be stitched. Second, we design an effective algorithm for extracting hidden attributes that are essential for the stitching process and for aligning values of those attributes across tables to synthesize new columns. We also assign meaningful names to these synthesized columns. Experiments on real world tables demonstrate the effectiveness of our approach.

1 Introduction

Tables on the Web have been recognized as an important source of structured data and have attracted a number of research efforts from both academia and industry [Cafarella *et al.*, 2008b; Gupta and Sarawagi, 2011; Wang *et al.*, 2012]. All of those works extract tables individually, and focus on annotating the tables for applications such as visualization, search, and knowledge base enrichment. While understanding raw tables independently is important, there is even more value in consolidating individual tables.

As an example, consider the set of tables at the Public School Review Site[1]. They record the statistics of schools all over the USA, as shown in Figure 1. For human readability, the site designers have fragmented the information into smaller tables where each table corresponds to a subset of the schools. However, a user might be interested in a different organization of the schools, such as finding all schools with over 500 students. Without a holistic view of the original table of all schools, such a task is nearly impossible to accomplish. Furthermore, realizing that these tables are part of a bigger whole can enable a table search engine to provide much more advanced utilities such as visualizing the schools by states or school types.

This example demonstrates the power of *table synthesis*, i.e., combining raw tables on the Web to produce union tables that are *more valuable than the sum of those individual tables*. In this paper, we provide a first step to the solution of this novel problem. We consider the problem of *table stitching*: combining multiple tables on the *same* site in the same schema that can be considered parts of a larger union table. Even this first step raises challenging technical problems. First, we must correctly discover tables that can be unioned. Second, we need to stitch the tables properly. Specifically, simply concatenating raw tables together will lead to a table that contains inconsistent data. In the example site shown in Figure 1, schools from different locations may have the same name. To properly combine them, we must recover their locations from the page context (e.g., the page title, or text surrounding the table) and add a new column in the table to show this geographic information. In our example, the ideal stitching result is shown on the right panel of Figure 2.

In this paper, we describe algorithms for finding stitchable/unionable tables, recovering the hidden attributes, and assigning them meaningful column names. Our main contributions are the following:

- We introduce the table stitching problem whose goal is

*This work was done during the first author's internship at Google.

[1]http://www.publicschoolreview.com/

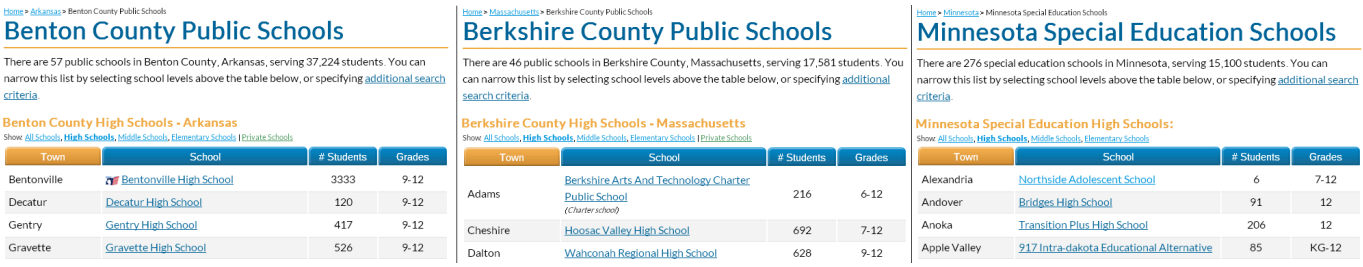

Figure 1: Screenshots of the Public School Review Website.

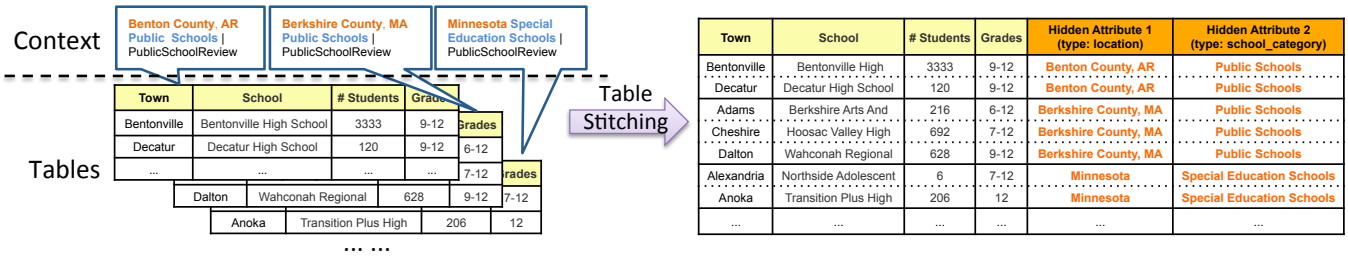

Figure 2: Table stitching. (Left) Raw tables and their context. (Right) An ideal synthesized table from the raw tables.

to structurally organizing individual tables with necessary context.

- We describe a segment-based multiple sequence alignment algorithm for extracting hidden table attributes from the table context, where each table context is considered as a word sequence. Given candidate segments from different heuristics as input, the algorithm seeks an optimal alignment of multiple sequences and determines the proper segmentations of those sequences.
- We describe techniques for giving the newly extracted attributes meaningful names.
- We present a set of experiments that show: 1) our method effectively extracts better hidden attributes than baseline methods; 2) a combination of candidate segments suggested by different heuristics works the best; and 3) we are able to automatically label extracted attributes with a set of predefined class labels with reasonable quality.

We define the terminology and present an overview of the table stitching problem in Section 2. Section 3 describes the core of our technique, namely extracting hidden attributes from table contexts. Together with the technique for assigning meaningful column names for those attributes (which we briefly describe within the experimental evaluation), this technique allows us to provide a first-cut solution for table synthesis. Experimental results are presented in Section 4, followed by a discussion on related work in Section 5. Finally, Section 6 concludes and proposes some future directions.

2 Overview

Assume a corpus of tables $\mathcal{T}$ extracted from the same site. Our goal is to build a pipeline for stitching tables in $\mathcal{T}$ into a set of union tables. This pipeline consists of three major components: the stitchable table identifier, the hidden attribute extractor and the table stitcher.

Stitchable Tables Identifier: To identify a set of stitchable tables from a corpus $\mathcal{T}$, we rely on the header h_T of a table T [2]. We define *stitchable tables* as follows:

Definition (Stitchable Tables): Two tables T_1 and T_2 are stitchable if their headers h_{T_1} and h_{T_2} are *semantically equivalent* regardless of the ordering, $h_{T_1} \equiv h_{T_2}$. Specifically, for all attribute names $h_{T_1}(i)$ in h_{T_1} there exists one and only one attribute name $h_{T_2}(j)$ in h_{T_2} that has the same meaning of $h_{T_1}(i)$ and the reverse also holds. Note that the stitchable property is transitive: if T_1 and T_2 are stitchable and so are T_2 and T_3, then T_1 and T_3 are stitchable.

We can potentially define semantic equivalence based on non-trivial notions such as synonymy, acronyms or many other semantic-preserving variations. For simplicity, however, we simply consider semantic equivalence as two sets of attribute names having exactly the same set of string values. Our stitching techniques are orthogonal to stitchable tables identification, thus we leave advanced identifiers for future work. Once identified, all stitchable tables are grouped together for stitching. Because of the transitive nature, each table can only belong to one stitchable group.

Hidden Attribute Extractor: For each stitchable group $G = \{T_1, \cdots, T_n\}$, our goal is to union the element tables without creating semantic ambiguities. As mentioned above, two schools from different counties could have exactly the same name. Information that can be used to disambiguate

[2] h_T is a set of string values that are the column names of T. In the example shown in Figure 2, they are "Town", "School", "# Students" and "Grades".

those rows are often hidden within the table context. The challenge, however, is that such information is usually presented as unstructured natural language context on the page—we need to extract, as precisely and concisely as possible, a set of structured hidden attributes m_T from such context to make each table distinguishable within the group. The set of hidden attributes m_T is represented as an ordered list of values, $\{m_1^T, \cdots, m_{M_G}^T\}$, where M_G is the number of hidden attributes for the group. Note that we are making the assumption that all tables within the same group have the same set of hidden attributes. If a particular attribute is missing from a table, we can simply assign it null for that table.

Table Stitcher: The final step is to stitch the tables in the same group. For each table T in a group G, its induced attributes m_T will be appended to each tuple in T. The augmented tables are then simply concatenated together. Those induced attribute values, however, are often difficult to understand and leverage by applications because of the lack of attribute names on the newly created columns. To enrich those hidden attributes, we leverage techniques inspired by [Venetis *et al.*, 2011], in which we match the values in the cells to a database of `isA` relations [Pasca and Van Durme, 2008]. If a significant number of values in a column get mapped to a common class in the `isA` database, we use the class name as the attribute name. We discuss the effectiveness of this simple approach using empirical results in Section 4.3.

3 Hidden Attribute Extraction

In this section, we focus on the main technical challenge of table stitching: extracting hidden attributes for the tables from unstructured context on the page. We discuss the sources of context (Section 3.1) and the techniques of extracting the implicit attributes (Section 3.2). We also describe a few heuristics to provide candidate segmentations that are crucial for the extraction (Section 3.3).

3.1 Sources of the Context

The hidden attributes may be embedded in different sources of context. For example,

- **Web page title**: A title is used to succinctly describe the main content of a Web page with necessary details. When the main content of a Web page is a table, its title is often useful for the extraction.
- **Text surrounding the table**: The description of a table often appears in text before the table[3]. We can extract such text by looking for the closest text node to the table in the page's DOM tree.

In addition, Web page URLs, HTML caption tags, and navigational menus on Web pages can also serve as useful sources of context, and our technique described below could similarly apply to those. As an initial study, we chose to focus on the title and surrounding text first, which are empirically the two most important sources.

[3]The description also appears after the table but in the initial experiments we empirically observe little useful text after the table.

3.2 Extraction by Segmentation and Alignment

The context of a table is simply a piece of unstructured text, thus the main challenge is to accurately identify the set of hidden attributes from those sequences and align them across all tables in the group so that the values are semantically coherent. Note that many attribute values are presented as phrases, instead of isolated tokens. For instance, the hidden attribute "Benton County, AR" have the most semantic value only when all three tokens are present: individual tokens such as "Benton" or "County" are not useful, while the token "AR" represents too large an area.

We tackle this task as a sequence labeling problem [Lafferty *et al.*, 2001] where the text is represented as a sequence of n tokens $T = \langle t_1, \ldots, t_n \rangle$. If a segment (i.e., a consecutive subsequence of tokens), represents a meaningful phrase, it will be labeled as a useful attribute value. However, our problem is different from traditional sequence labeling problem since the extracted segments for different tables in the same group need to be aligned so that they can be filled into the implicit columns of the resulting union table.

Our solution is inspired by a similar problem in computational biology, namely finding common genetic motifs from different DNA sequences [Gusfield, 1997]. To identify genetic mutations within a group of individuals, scientists compare those individuals' DNA sequences at the same time using the *Multiple Sequence Alignment* (MSA) technique. The context of our tables can be considered as the DNA sequences, and thus we can adapt the MSA technique to identify and align useful hidden attribute segments.

However, there are a few further challenges. First, MSA is not directly applicable. In DNA sequence alignment, the basic unit for alignment is individual nucleobases, namely A, T, C and G. In our case, the segments are the basic units, but we only have tokens as input. We have to solve the segmentation and the alignment problem holistically. Second, we do not have enough training data to apply any existing supervised segmentation method. It is extremely hard to manually label table context data from the whole Web since each site has different characteristics. Therefore, we turn to unsupervised methods. In particular, we design a suite of segmentation heuristics. Each heuristic captures some characteristics of the hidden attributes, but may miss others. However, the heuristics work surprisingly well collectively. More concretely, to select the segments, we adopt the following strategy. If segments by the a particular heuristic tend to align across sequences, then that particular heuristic is more likely to be correct. We next introduce the segment-based multiple sequence alignment method. (The heuristics for generating candidate segments will be detailed in Section 3.3.)

Before diving into the details of alignment of n sequences, we first look at the pairwise case when $n = 2$ (Algorithm 1). We have two sequences T_1 and T_2 (we abuse the symbol T here to also represent table context). Their candidate segments (S_1 and S_2 respectively, where each element segment is represented by begin and end token positions) are generated by several heuristic. In addition, empty segments are added to allow null values (i.e., gaps) for alignment. Similar to other dynamic programming algorithm, we divide the whole problem into subproblems. How well the segments

from each sequence align depends on how well these two segments match *and* how well the subsequences immediately before each segment are aligned. Let $|T_1| = n_1$ and $|T_2| = n_2$. We maintain a chart C of a size $(n_1+1)\cdot(n_2+1)$, where each chart entry $C(i,j)$[4] stores the score for the best alignment between the subsequences $T_1^{1\cdots i}$ and $T_2^{1\cdots j}$, as well as the last aligned segment for each subsequence. The algorithm runs two outer loops ranging from the smallest subproblems to the final whole problem. At each subproblem (i,j), we enumerate all pairs of the candidate segments that end with token T_1^i and T_2^j respectively. Note that $S_l^i, l \in [1,2]$, is defined as a set of candidate segments from S_l that end at T_l^i. For each pair, a segment matching score is computed as follows:

$$score(s_1, s_2) = \begin{cases} \lambda_h & \text{if both } s_1 \text{ and } s_2 \text{ are generated} \\ & \text{by the same heuristic } h; \\ \lambda_{gap} & \text{if } s_1 \text{ or } s_2 \text{ is an empty segment;} \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

The sum of this segment matching score and the best alignment score of the immediate previous subproblem is used to update the chart entry.

$$C(i,j) \leftarrow \max(C(i,j), score(s_1,s_2) + C(i-|s_1|, j-|s_2|)) \tag{2}$$

In the end, we can extract the aligned segments by tracing back from the chart entry $C(n_1, n_2)$.

Input: Two sequences of tokens T_1 and T_2 of size n_1 and n_2 and two sets of candidate segments S_1 and S_2 respectively.
Output: The best alignment of segments in T_1 and T_2.
Initialization: A chart C of size $(n_1+1)\cdot(n_2+1)$.
for $i \leftarrow 0$ **to** n_1 **do** $C(i,0) = i \times \lambda_{gap}$;
for $j \leftarrow 1$ **to** n_2 **do** $C(0,j) = j \times \lambda_{gap}$;
for $i \leftarrow 1$ **to** n_1, $j \leftarrow 1$ **to** n_2 **do**
  for $s_1 \in S_1^i$, $s_2 \in S_2^i$ **do**
    Update the chart at $C(i,j)$ according to Eq. 2 ;
  end
end

Algorithm 1: Pairwise Segment Alignment

In principle, we can use dynamic programming in a similar fashion for multiple ($n > 2$) sequences to compute an optimal sum-of-pairs score, where the optimal alignment will have the best score summing over all pairs of pairwise alignment scores. Unfortunately, these computations are exponential in the number of sequences. Previous literature has proved that finding the optimal MSA when the number of sequences is a variable is NP-complete [Wang and Jiang, 1994]. Therefore, we approximate the solution by iterative pairwise alignment (similar to [Barzilay and Lee, 2003]). In particular, we maintain a profile of current pairwise alignment which can also be viewed as a sequence. Each element of this pseudo-sequence is, instead of a single token, a distribution of different segments in this alignment slot. When aligning with another original sequence, the algorithm remains the same except that the `score` function is overloaded for what we call a **profile slot**. Specifically, a profile slot ps is a set of segment-probability pairs, $\{(s_i, p_i)\}$, and its alignment score with a segment s_j is defined as a weighted sum of the scores between each segment in the slot and the segment s_j:

$$score(ps, s_j) = \sum_{(s_i,p_i)\in ps} p_i \cdot score(s_i, s_j) \tag{3}$$

Finally we can recover the aligned segments from the profile by reading each slot. We further post-process the results by filtering out useless segments if they do not provide distinguishing information for the stitched tables. Specifically, a slot will be removed if it meets two conditions: 1) all the segments of this slot have the same value; and 2) the value cannot be used as hidden attribute labels. The first condition rules out the slots such as preprosition words or website names that are not specific to the tables. However, a constant value sometimes could be used as a hidden attribute label, e.g. a slot with a constant value "Area:" would be an appropriate attribute label for the slot next to it. To prevent a reasonable attribute label from being removed, we further check if the value is present in a pre-existing attribute label database where the string values are ordered by the numbers of their appearances in the table headers from a corpus of millions of WebTables ([Cafarella *et al.*, 2008b][5]).

3.3 Heuristics for Candidate Segments

We now discuss how to generate candidate segments via several heuristics. We adopt three diverse heuristics that treat the text in different ways ranging from purely syntactic to semantic interpretations.

Punctuation/Tag Separators: When pieces of text are not organized in a grammatical sentence on Web pages, they are either separated by obvious punctuations, e.g., commas, vertical bars, or highlighted by different HTML styles, e.g. font colors, font size. In the first case, we use the segments of tokens between two punctuation marks. For style highlights, the HTML tags are sufficient for the segmentation.

Longest Common Subsequences (LCS): Assuming some contextual texts are automatically generated using templates, another heuristic is first detecting the common segments from the context and use them to separate the larger sequence. The remaining segments that have different values across sequences are extracted as the hidden attribute segments. This problem has long been approached as Longest Common Subsequences where each subsequence is a common segment in our context as a separator. We start by comparing a pair of sequences of tokens, which can be efficiently solved by dynamic programming [Bergroth *et al.*, 2000]. The method proceeds by iteratively and greedily comparing to the next sequence for LCS. Note that an LCS problem can be seen as a degraded MSA problem with the tokens being the aligning elements and the scoring function being binary on string matches (1 for the matches).

[4] We use $C(i,j)$ to represent both the chart entry and the alignment score for that entry.

[5] We re-build the attribute database from our own Web table corpus and only consider the top 5K frequent attribute names.

Wikification: Wikification [Milne and Witten, 2008; Ratinov *et al.*, 2011] is a technique of linking words or phrases in any text to a corresponding Wikipedia article. If a segment can be wikificated, it will likely to be meaningful and useful for understanding the table. We applied a homegrown wikification tool to all the contextual text and extracted the segments identified as Wikipedia entities as candidate segments for alignment.

4 Experimental Evaluation

We present experiments demonstrating the effectiveness of our techniques for hidden attribute extraction and alignment (Section 4.2), followed by experiments on labeling the extracted attribute columns (Section 4.3).

4.1 Data Set

We obtained a corpus of 130 million Web tables that were filtered from a collection of 14 billion raw HTML tables crawled from the Web. From the corpus, we performed the simple stitchable table identification by grouping the tables based on their sites and the automatically detected header rows. For the experiments, we sampled 20 large groups, each of which has more than 1000 tables, from 10 different websites[6]. For each group, we further sampled 10 individual tables for evaluation, which is conducted based on golden attribute values obtained through human evaluators judging from the table context.

4.2 Hidden Attribute Extraction and Alignment

In this section, we investigate the quality of hidden attributes extraction and alignment. We perform the quality analysis from two perspectives, cell-wise accuracy and column-wise accuracy, where the former evaluates the extraction while the latter evaluates the alignment. We also empirically show that different heuristics for segmentation are complementary and work the best in concert.

Methodology: We evaluate our approach by comparing different combinations of candidate segments as described in Section 3.3. The parameters in Eq. 1 are determined via a grid search where each parameter can vary from 0.1 to 1.0 in an increment of 0.1. We report the performance numbers via a leave-one-out experiment.

Evaluation Metrics: For cell-wise accuracy, we evaluate how accurate the identified segments are. Adopting the standard Precision/Recall/F1 measures, we deem a predicted segment as a true positive (TP) if the prediction matches a labeled segment and a false positive (FP) otherwise. A false negative (FN) is a labeled segment that none of the predictions matches. Precision and recall are computed using $\frac{\#TP}{\#TP+\#FP}$ and $\frac{\#TP}{\#TP+\#FN}$, respectively, and F1 is the harmonic mean of the two. The same metrics apply to column-wise accuracy evaluation except that we only deem a column correct when all the segments (across rows from different individual tables) from a column are correct.

[6]Examples: www.century21.com, www.britishcycling.org.uk.

Experimental Results

Cell-wise: The cell-wise performance is reported in Table 1. As expected, alignment with candidate segments generated by a single heuristic (`SEP` or `LCS`) works moderately on identifying hidden attributes correctly, however, the coverage leaves much to be desired. The best strategy involves all three segmentation heuristics, `SEP+LCS+WK`, which achieves an F1 relatively 15% higher than the second best, `LCS+WK`. This large gain of F1 measure is obtained by significantly improving the recall: `SEP+LCS+WK` has a relative 35%+ higher recall than other strategies. This is consistent with our expectation because leverage all available heuristics brings in a large and diverse set of segmentation candidates.

Segments	Precision	Recall	F1
SEP	0.458	0.260	0.332
LCS	0.630	0.478	0.543
SEP+LCS	0.551	0.484	0.516
LCS+WK	0.650	0.516	0.575
SEP+LCS+WK	0.627	0.703	0.663

Table 1: Performance on hidden attribute extraction with different combinations of segmentation heuristics: `SEP` is the separator-based heuristics in Section 3.3, `LCS` is the Longest Common Subsequences heuristics, and `WK` refers to the wikification-based heuristics.

More importantly, we note that the candidate segments from syntactic (`SEP` and `LCS`) and semantic (`WK`) heuristics are complementary. Comparing `SEP+LCS` against `SEP+LCS+WK` or `LCS` against `LCS+WK`, both precision and recall are improved thanks to the addition of the Wikification-based segments. The syntactic heuristics work well in many cases. For instance, assume two sequences "Location: Seattle, WA" and "Location: Portland, OR". The segment "Location:" is identified as the common segment and therefore the segments "Seattle, WA" and "Portland, OR" are correctly identified as potential hidden attributes. However, in the example of "Springfield High School" and "Jacksonville Elementary School", the syntactic heuristic will naively recognize the segment "School" as the common separator because it largely ignores the semantic meaning of the whole phrase, while Wikification-based segmentation will recognize "Jacksonville Elementary School" as a single entity. In particular, the Wikification heuristic prevents over-segmenting a phrase, as well as helps segmenting large text chunks into meaningful phrases where punctuation is not available. For example, "American Airline (AA) #1430" is a single segment by syntactic heuristics while a semantic heuristics can break it up into the airline name and the flight number.

We also observe that surrouding text are comparatively harder for extraction than titles. It is partly because the collection of surrounding text can make errors (e.g. absense of relevant text in the closest DOM node).

Column-wise: We further examine if correct hidden attributes are created column-wise. The cell-wise evaluation shows the performance of sequence labeling while the column-wise evaluation measures the quality of the entire alignment. We evaluated only the best strategy `SEP+LCS+WR` on the same set of Precision/Recall/F1 num-

bers (the other strategies are significantly worse). In summary, our method generated 62 attribute columns across 20 table groups, 24 of which are column-wise correct (i.e. every cell of the column matches the labels), and 31 labeled columns were missed, resulting in a precision/recall/F1 at 0.387 / 0.436 / 0.410. If we relax the correctness condition by allowing one wrong cell in a column, the F1 measure will be improved to 0.547. Looking at the table groups individually, 7 out of 20 groups have a perfect match between the predictions and the human labels. We note that the content of the wrongly generated columns due to segmentation errors could still be useful in the searching scenarios.

4.3 Hidden Attribute Column Label Prediction

In this section, we examine the effectiveness of automatically labeling the types for automatically extracted attribute columns, using a straight-forward column labeler based on the cell values. Given a provided `isA` database as described in Section 2, each cell value is matched to a list of zero or more types. A type is assigned to a column if and only if at least $t\%$ of its cell values have that type. Each predicted label is then manually marked as `vital`, `ok`, or `incorrect` depending on how accurate and appropriate the label is for the column, as done in [Venetis *et al.*, 2011]. To compute precision, a predicted label is scored 1 if it is marked as `vital`, 0.5 if marked as `ok`, or 0 otherwise. On average, there are five relevant labels for a given column. Thus, for fairness, if there is no relevant label marked for a given column, we assume that there are five missed relevant labels for computing the recall.

We vary the threshold t from 0.05 to 1 in an increment of 0.05 to draw a precision-recall curve (shown in Figure 3). We observe a performance comparable to [Venetis *et al.*, 2011]. The precision ranges from 0.4 to 1.0 and the maximum recall is 0.89. We are thus confident that this provides a good overview of the hidden attribute columns.

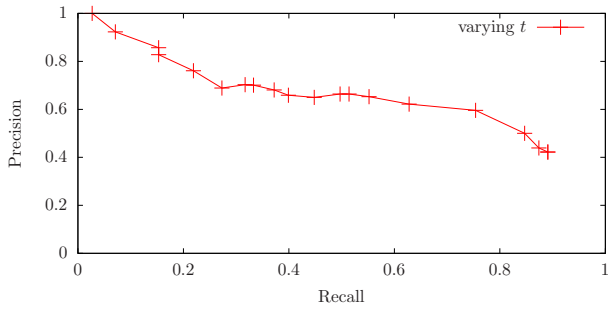

Figure 3: Precision/Recall diagram of the label predictions for the hidden attribute columns.

5 Related Work

There have been several studies on extracting tables from the Web. [Gatterbauer and Bohunsky, 2006] extract tables from Web pages based on their DOM nodes' visual positions. [Cafarella *et al.*, 2008b] proposed the initial WebTables vision for web-scale table extraction, implementing a mix of hand-written detectors and statistical classifiers that identified 154 million high-quality relational style tables from a raw collection of 14.1 billion tables on the Web. [Elmeleegy *et al.*, 2009] split lists on Web pages into multi-column tables in a domain-independent and unsupervised manner.

Some previous works tried to discover and leverage the relationship between individual tables. [Pimplikar and Sarawagi, 2012] answers a query from millions of tables by identifying a table's relevance to the query and mapping the columns of relevant tables to the query's key words. Table synthesis is also connected to finding related tables [Das Sarma *et al.*, 2012] and yet very different. The latter is only concerned with finding tables that are relevant in some way, such as being about the same topic, even when those tables are entirely not stitchable.

The Octopus System [Cafarella *et al.*, 2009] includes a context operator that tries to identify additional information about the table on a Web page. The operator was implemented by ranking a keyword list according to the Tf/Idf scores. Our work differs in that our extracted attribute values are represented as meaningful phrases instead of isolated keywords. Moreover, we align them in columns, providing a structured view across the tables instead of orderless lists of keywords.

Lastly, the hidden value extraction algorithm is based on Multiple Sequence Alignment (MSA), originally designed to compare DNA sequences [Gusfield, 1997]. Also, [Barzilay and Lee, 2002; 2003] applied MSA to lexicon acquisition for a text generation system and to find paraphrasing patterns respectively. In contrast, instead of being given two sequences of fixed symbols for alignment, we need to determine the segments of tokens on the fly and simultaneously align those segments. For that, the classic sequence alignment algorithm is extended for our scenario by incorporating candidate segments from various heuristics. The hidden value extraction is also related to wrapper induction [Crescenzi and Mecca, 2004]. The main differences lie in that our approach does not heavily rely on HTML tags and that our work can be applied to extractions on different websites.

6 Conclusion

We proposed an effective solution for stitching tables on the same site into a meaningful union table, as the first step towards addressing the challenge of synthesizing tables on the Web. Specifically, our solution identifies tables that are stitchable, infers necessary yet minimal set of hidden attributes from the context, unions the tables into a single table with proper attribute values for disambiguation, and finally provides the context with reasonable labels. Our experiments over a corpus of real world tables demonstrate the effectiveness of our proposed solution over various aspects of the stitching process. We believe table synthesis will enable users to explore the harvested structured data with more powerful capabilities.

In future work, we plan to design algorithms to improve the stitchable table identifier, extract hidden attributes from other sources (e.g. URLs of the Web pages) and investigate the relationship between the table content and induced hidden attributes.

Acknowledgments

We would like to thank Shirley Zhe Chen and Spiros Papadimitriou for early exploration and helpful discussion on this project.

References

[Barzilay and Lee, 2002] R. Barzilay and L. Lee. Bootstrapping lexical choice via multiple-sequence alignment. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 164–171. Association for Computational Linguistics, 2002.

[Barzilay and Lee, 2003] R. Barzilay and L. Lee. Learning to paraphrase: An unsupervised approach using multiple-sequence alignment. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 16–23. Association for Computational Linguistics, 2003.

[Bergroth *et al.*, 2000] L. Bergroth, H. Hakonen, and T. Raita. A survey of longest common subsequence algorithms. In *String Processing and Information Retrieval, 2000. SPIRE 2000. Proceedings. Seventh International Symposium on*, pages 39–48. IEEE, 2000.

[Cafarella *et al.*, 2008a] M.J. Cafarella, A. Halevy, D.Z. Wang, E. Wu, and Y. Zhang. Webtables: exploring the power of tables on the web. *Proceedings of the VLDB Endowment*, 1(1):538–549, 2008.

[Cafarella *et al.*, 2008b] M.J. Cafarella, A.Y. Halevy, Y. Zhang, D.Z. Wang, and E. Wu. Uncovering the relational web. *WebDB*, 2008.

[Cafarella *et al.*, 2009] M.J. Cafarella, A. Halevy, and N. Khoussainova. Data integration for the relational web. *Proceedings of the VLDB Endowment*, 2(1):1090–1101, 2009.

[Crescenzi and Mecca, 2004] Valter Crescenzi and Giansalvatore Mecca. Automatic information extraction from large websites. *Journal of the ACM (JACM)*, 51(5):731–779, 2004.

[Das Sarma *et al.*, 2012] A. Das Sarma, L. Fang, N. Gupta, A. Halevy, H. Lee, F. Wu, R. Xin, and C. Yu. Finding related tables. In *Proceedings of the 2012 international conference on Management of Data*, pages 817–828. ACM, 2012.

[Elmeleegy *et al.*, 2009] H. Elmeleegy, J. Madhavan, and A. Halevy. Harvesting relational tables from lists on the web. *Proceedings of the VLDB Endowment*, 2(1):1078–1089, 2009.

[Gatterbauer and Bohunsky, 2006] W. Gatterbauer and P. Bohunsky. Table extraction using spatial reasoning on the css2 visual box model. In *PROCEEDINGS OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE*, volume 21, page 1313. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006.

[Gupta and Sarawagi, 2011] Rahul Gupta and Sunita Sarawagi. Joint training for open-domain extraction on the web: Exploiting overlap when supervision is limited. In *WSDM*, 2011.

[Gusfield, 1997] D. Gusfield. *Algorithms on strings, trees and sequences: computer science and computational biology*. Cambridge University Press, 1997.

[Lafferty *et al.*, 2001] J. Lafferty, A. McCallum, and F.C.N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.

[Limaye *et al.*, 2010] G. Limaye, S. Sarawagi, and S. Chakrabarti. Annotating and searching web tables using entities, types and relationships. *Proceedings of the VLDB Endowment*, 3(1-2):1338–1347, 2010.

[Milne and Witten, 2008] D. Milne and I.H. Witten. Learning to link with wikipedia. In *Proceedings of the 17th ACM conference on Information and knowledge management*, pages 509–518. ACM, 2008.

[Pasca and Van Durme, 2008] M. Pasca and B. Van Durme. Weakly-supervised acquisition of open-domain classes and class attributes from web documents and query logs. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics (ACL-08)*, pages 19–27, 2008.

[Pimplikar and Sarawagi, 2012] Rakesh Pimplikar and Sunita Sarawagi. Answering table queries on the web using column keywords. In *Proc. of the 38th Int'l Conference on Very Large Databases (VLDB)*, 2012.

[Ratinov *et al.*, 2011] L. Ratinov, D. Roth, D. Downey, and M. Anderson. Local and global algorithms for disambiguation to wikipedia. In *Proceedings of the Annual Meeting of the Association of Computational Linguistics (ACL)*, 2011.

[Venetis *et al.*, 2011] P. Venetis, A. Halevy, J. Madhavan, M. Paşca, W. Shen, F. Wu, G. Miao, and C. Wu. Recovering semantics of tables on the web. *Proceedings of the VLDB Endowment*, 4(9):528–538, 2011.

[Wang and Jiang, 1994] L. Wang and T. Jiang. On the complexity of multiple sequence alignment. *Journal of computational biology*, 1(4):337–348, 1994.

[Wang *et al.*, 2012] Jingjing Wang, Haixun Wang, Zhongyuan Wang, and Kenny Qili Zhu. Understanding tables on the web. In *ER*, pages 141–155, 2012.