TEGRA: Table Extraction by Global Record Alignment

Xu Chu[1] *, Yeye He[2], Kaushik Chakrabarti[2], Kris Ganjam[2]
[1]University of Waterloo, Waterloo, Canada
[2]Microsoft Research, Redmond, USA
[1]x4chu@uwaterloo.ca
[2]{yeyehe, kaushik, krisgan}@microsoft.com

ABSTRACT

It is well known today that pages on the Web contain a large number of content-rich relational tables. Such tables have been systematically extracted in a number of efforts to empower important applications such as table search and schema discovery. However, a significant fraction of relational tables are *not* embedded in the standard HTML table tags, and are thus difficult to extract. In particular, a large number of relational tables are known to be in a "list" form, which contains a list of clearly separated rows that are not separated into columns.

In this work, we address the important problem of automatically extracting multi-column relational tables from such lists. Our key intuition lies in the simple observation that in correctly-extracted tables, values in the same column are *coherent*, both at a syntactic and at a semantic level. Using a background corpus of over 100 million tables crawled from the Web, we quantify semantic coherence based on a statistical measure of value co-occurrence in the same column from the corpus. We then model table extraction as a principled optimization problem – we allocate tokens in each row sequentially to a fixed number of columns, such that the sum of coherence across all pairs of values in the same column is maximized. Borrowing ideas from $A^\star$ search and metric distance, we develop an efficient 2-approximation algorithm. We conduct large-scale table extraction experiments using both real Web data and proprietary enterprise spreadsheet data. Our approach considerably outperforms the state-of-the-art approaches in terms of quality, achieving over 90% F-measure across many cases.

1. INTRODUCTION

Relational tables on the Web have long been established as a rich source of structured data. Early studies suggest that the number of meaningful relational tables on the Web is well over a hundred million [10, 26], and is thus considered to be by far the largest relational data repository [10]. The sheer number of tables has spawned many interesting applications including table search [1, 2], table integration [9, 27], and knowledge discovery [26] among many others.

Figure 1: An example relational table in HTML list

All these useful applications hinge on the successful extraction of high-quality relational tables. While many relational tables on the Web are embedded in *HTML table* tags, and are thus easy to extract (since both row boundaries and column boundaries are clearly marked), a significant fraction of tables do *not* use HTML table tags. In particular, a large number of relational tables are in *HTML list* tags, which provide a set of clearly separated rows, but do not segment the rows into columns. Figure 1 shows one such example list containing a relational table with city name, state name, and population of cities in New England.

It is estimated in [16] that the number of useful relational tables that can be extracted from HTML lists is at least tens of millions (a number consistent with our analysis to be discussed later). Tables in lists thus represent a significant source of relational tables. Extracting such tables brings obvious benefits for downstream applications. For example, a table search system [1, 2] powered by an enhanced table corpus extracted from lists is more likely to return relevant tables (e.g., the table in Figure 1 can be returned for a query "top New England city population").

The pioneering work from Google, named `ListExtract` [16], is the only work that specifically targets the problem of extracting tables from lists on the Web. The authors rightfully point out the challenges therein – column delimiters in lists are implicit, which can be any special characters. Delimiters can also vary from one list to another, or even within the same list. In Figure 1, for example, a comma (",") and a semi-colon (":") are used as column delimiters, yet the second comma (",") is not a delimiter. In addition, white-spaces (" ") is often used as delimiters, as in the additional example in Figure 2 (which should parse into a three-column table shown in Figure 3). In Figure 1, however, white-spaces should not be used as delimiters. The bottom line is that any special characters or punctuation can be used as column de-

*Work done at Microsoft Research.

SIGMOD'15, May 31–June 4, 2015, Melbourne, Victoria, Australia.

http://dx.doi.org/10.1145/2723372.2723725.

l_1	Los Angeles California United States
l_2	Toronto Canada
l_3	New York City New York USA

Figure 2: An example input list L

t_1	Los Angeles	California	United States
t_2	Toronto		Canada
t_3	New York City	New York	USA

Figure 3: An extracted output table T

limiters, which greatly complicates the extraction problem. [1]

Previous approaches. The authors in [16] were the first to recognize the value of extracting tables from We lists, and proposed an approach called `ListExtract`. To date, [16] is the only published approach specifically addressing the problem of extracting tables from general Web lists fully automatically (other related methods will be reviewed in Section 6).

At a high level, the `ListExtract` approach works in three steps: (1) It utilizes various signals, such as a language model, to greedily identify token subsequences in each line that are likely to be individual cells (e.g., "New York" and "USA"), so that each line is independently split into multiple cells. (2) It counts the number of segmented cells in each line and use the majority vote to guess the correct number of columns, k, in the output table. Records that are not split into k columns in the previous step are re-split to produce exactly k columns. (3) Adjustments are made for values that appear inconsistent to produce the final output. A more detailed discussion of `ListExtract` can be found in Appendix A.

We observe that `ListExtract` makes *local* splitting decisions in each line *independently* early in the process, regardless of segmentations in other lines. Once such local decisions are made, it becomes difficult to recover from these decisions that are in fact suboptimal globally. For example, in l_3 of Figure 2, the language model may be very confident that "New York" is more likely to be a cell entry than "New York City" because of the popularity of the name. However, once "New York" is greedily split into a cell, it becomes very difficult to segment subsequent tokens starting with "City" (e.g., "City", "City New", or "City New York") to align them well with values from other rows in the same column. Such local greedy decisions negatively affect the quality of the extraction.

Highlights of our approach. In light of these issues, in this work we propose a principled approach called Table Extraction by Global Record Alignment (TEGRA), which models table extraction as a global optimization problem. We observe that in a correctly extracted table, cell values in the same column should be coherent, either semantically (e.g., "Los Angeles" and "Toronto" are coherent because they are in the same domain and co-occur in the same column frequently), or syntactically (e.g., values should have similar token length, etc.). Using a Web table corpus with over 100 million tables, we quantify semantic coherence by statistical value co-occurrence in the same column (e.g., "Los Angeles" and "Toronto" co-occur much more frequently than mere coincidence in the corpus, thus their coherence is strong). Using the notion of coherence, we formulate table extraction as an optimization problem – we sequentially allocate tokens to a fixed number of columns, so that the sum of coherence score across all pairs of values in the same column can be maximized.

[1]The work in [16] provides additional HTML lists that are excellent motivating examples.

While this is a natural formulation that captures the "goodness" of a well-formatted table, the formal problem is intractable in general. We design a mechanism to decompose the objective function into more tractable components while guaranteeing 2-approximation. We further develop an efficient algorithm utilizing ideas from A* search that can prune away unpromising segmentations without affecting quality.

TEGRA has a number of salient features. First, by using coherence as the objective function in our formulation, we perform *global* optimization, where splitting decisions in different lines are optimized *holistically* to ensure the best alignment.

Second, our coherence score takes advantage of a large table corpus with over 100 million tables. By using a data-driven approach to measure the compatibility of two values in the same column ("Los Angeles" and "Toronto"), it easily generalizes to a wide variety of domains, which works well for both public Web data as well as proprietary enterprise data, as will be shown in our experiments. In comparison, previous research on unsupervised record segmentation [3, 14,28] mostly relies on a specific reference table or knowledge base that need to match the target table of interest, and as we will show, cannot handle general tables on the Web.

Third, in order to scale to millions of lists on the Web and extract tables automatically, our approach is designed to be fully unsupervised, for user input can be too expensive at this scale. Note that in our problem, we need to extract content with heterogeneous schemas from *every* lists. This is in contrast to techniques such as wrapper induction [4, 12, 17], that utilize training examples to extract targeted relations with a specific schema.

Finally, although the intended application is offline relation table extraction (the *offline unsupervised variant*), our approach is general and capable of handling an interactive online scenario for ad-hoc table extraction (the *online supervised variant*), where a certain amount of user feedback is given in the form of segmentations for a few example rows to further improve the quality of tables extracted.

Contributions. We make the following contributions.

- We model table extraction as a principled optimization problem that captures the conceptual goodness of a table. We show experimentally that this objective function strongly correlates with the quality of extracted tables.
- We develop an efficient 2-approximation algorithm utilizing ideas from $A^\star$ search and a mechanism to decompose the objective function.
- We demonstrate that our approach can also be adapted to handle user supervision in the same framework.
- We conduct extensive experiments using real datasets from both the public Web and a proprietary enterprise. TEGRA achieves high accuracy and outperforms the state-of-the-art in all scenarios we evaluated.

2. PROBLEM STATEMENT

In this section, we formally define the problem of *table extraction from lists*. We start by defining what table segmentation or table extraction is[2].

2.1 Definition of Segmentation

Let *tok* be a tokenization function that splits an unsegmented line l into a sequence of tokens $tok(l)$, based on a set of user defined delimiters (e.g., white spaces, commas, etc.). We use $l[w]$ to index the w^{th} token, and use $l[i \dots j], i \leq j$,

[2]We use both table segmentation and extraction interchangeably when the context is clear.

to represent a consecutive subsequence of tokens starting at $l[i]$ and ending at $l[j]$. We define a *segmentation* t of line l as follows.

Definition 1. *An m-column segmentation t of line l is defined as $t = (l[i_1 \dots j_1], \dots, l[i_m \dots j_m])$, where $i_1 = 1, j_m = |l|$, and $i_{k+1} = j_k + 1, \forall 1 \leq k \leq m-1$. Denote by $s_m(l)$ the domain of all possible m-column segmentations of l.*

The notion of segmentation is quite natural – it is a set of m subsequences of tokens that collectively covers all tokens in l. We intentionally make a distinction by denoting an unsegmented line as l and a segmented tuple/record[3] as t.

Example 1. *Consider an unsegmented line l_1 in our running example in Figure 2. Using white spaces as delimiters, l_1 can be split into a sequence of tokens as follows.*

l_1 :	Los	Angeles	California	United	States

A possible segmentation of l_1 is $t = (l[1 \dots 2], l[3 \dots 3], l[4 \dots 5])$

t :	Los Angeles	California	United States

Finally, a table segmentation for a list of lines L is simply the union of segmentations of all lines in L.

2.2 The Goodness of Segmentation

There are many ways to segment a line, and exponentially more to segment a list into a table. Among all these options, we need a criteria to measure the goodness of a segmented table in order to guide the algorithm to pick the best segmentation. Intuitively, when a human looks at a table, if values in the same column are not *coherent*, then he would judge the segmentation as "bad". For example, if a person's name "John Smith" and a city name "New York City" are in the same column, the segmentation would be judged as "bad" because **semantically** these values are not in the same conceptual domain. Similarly, if a numerical value "159.3" and a city name "New York City" are aligned in the same column, or if a long string value "New York City New York" and a short string "Toronto" are aligned in the same column, then the alignment is also likely to be bad, because we expect values in the same column of a correctly-segmented table to be **syntactically** similar (e.g., similar number of tokens, similar value types, etc.).

Mimicking this human intuition, we say a segmented table is good if it is overall coherent. For technical reasons, in this work we use the notion of *distance*, which is the reverse of coherence – that is, pairs of cell values with *high* coherence (similarity) have *low* distance (dis-similarity), and vice versa. With this, we can equivalently say that a segmented table is good if its overall distance is low.

2.3 Quantify Goodness: Distance Functions

In order to quantify the overall distance of a table, we first define the distance between two cell values. Given two string values s_1 and s_2, we use $d(s_1, s_2)$ to denote the *distance* (incoherence) score of s_1 and s_2. As motivated above, $d(s_1, s_2)$ should be a combination of *syntactic distance*, $d_{syn}(s_1, s_2)$, and *semantic distance*, $d_{sem}(s_1, s_2)$. Namely,

$$d(s_1, s_2) = \alpha\, d_{syn}(s_1, s_2) + (1 - \alpha)\, d_{sem}(s_1, s_2) \quad (1)$$

where α controls the relative importance of the two components. By default, we set α to be 0.5.

In this work, we focus on distance $d(s_1, s_2)$ that satisfies the natural properties used in metric distance, namely, non-negativity, symmetry, and triangular inequality.

- Non-negativity: $d(s_1, s_2) \geq 0$
- Symmetry: $d(s_1, s_2) = d(s_2, s_1)$
- Triangle inequality: $d(s_1, s_2) + d(s_1, s_3) \geq d(s_2, s_3)$

[3]Tuple and record are used interchangeably for a segmented row in the paper.

2.3.1 Semantic Distance

As discussed above, semantic distance $d_{sem}(s_1, s_2)$ should capture the likelihood that s_1 and s_2 are in the same semantic domain (e.g., "New York City" and "Toronto"). While specialized Knowledge Bases (KBs) capture some of these semantic information, they are of limited domain coverage and/or entity coverage (let alone name variations, etc.). Since our goal is to extract tables from millions of general lists on the Web regardless of domain, KBs are unsuitable as a universal way to define distance because it cannot possibly cover all entities that can be found on the Web (we will explore this in our experiments).

We propose a data-driven way to define semantic distance that generalizes to all values. Specifically, using a table corpus of over 100 million tables crawled from the Web, we measure the statistical co-occurrence of s_1 and s_2 in the same column. This embodies our intuition that values in the same semantic domain ("New York City" and "Toronto") should co-occur frequently in the same column.

In particular, let $C(s_1)$ and $C(s_2)$ be the set of Web table columns that s_1 and s_2 occur in, respectively. If $C(s_1) \cap C(s_2)$ is large relative to $C(s_1)$ and $C(s_2)$, then s_1 and s_2 are likely to be semantically coherent (low semantic distance).

In this work, we use a probabilistic measure called *pointwise mutual information* [11], $\text{PMI}(s_1, s_2)$, defined as follows. Let N be the total number of columns. Then $p(s_1) = \frac{|C(s_1)|}{N}$, $p(s_2) = \frac{|C(s_2)|}{N}$, are the probabilities of seeing s_1, s_2 respectively, and $p(s_1, s_2) = \frac{|C(s_1) \cap C(s_2)|}{N}$ is the probability of seeing (s_1, s_2) together in the same column. PMI is defined as:

$$\text{PMI}(s_1, s_2) = \log \frac{p(s_1, s_2)}{p(s_1)p(s_2)}$$

Example 2. *Let s_1 = Canada, and s_2 = Republic of Korea. Suppose $N = 100M$ (there are a total of 100M columns), $|C(s_1)| = 1000$, $|C(s_2)| = 500$, and $|C(s_1) \cap C(s_2)| = 300$ (individually, the two strings occur 1000 and 500 times respectively; together they co-occur 300 times). It can be calculated that $PMI(s_1, s_2) = 4.77 > 0$, strongly indicating that they are semantically related (even though syntactically s_1 and s_2 are very different, by the number of tokens, etc.)*

Note that PMI actually measures coherence instead of distance. We apply the following normalization and transformation to ensure that the resulting semantic distance satisfies triangle inequality[4].

$$d_{sem}(s_1, s_2) = 0.75 - 0.25\text{NPMI}(s_1, s_2)$$

where $\text{NPMI} \in [-1, 1]$ is the normalized PMI, defined as:

$$\text{NPMI}(s_1, s_2) = \frac{\text{PMI}(s_1, s_2)}{-\log p(s_1, s_2)}$$

Discussions. In principle other metric distances such as Jaccard distance, Angular distance (the metric version of Cosine similarity) can also be used to define $d_{sem}(s_1, s_2)$. We choose NPMI because it is a symmetric measure that is robust to asymmetric sets. Appendix H gives more discussion on this, as well as our experience of using Jaccard (which also produces decent results).

2.3.2 Syntactic Distance

The main motivation of using syntactic distance is that not all strings are semantically meaningful. For example, if

[4]This is by virtue of the fact that $d_{sem} \in [0.5, 1]$ after transformation of NPMI. When other metric distances are used in place of NPMI the triangle inequality is naturally respected.

s_1 = SKU-926434 and s_2 = SKU-09393. There is unlikely sufficient statistical co-occurrence to justify that they are semantically coherent.

At a high level, we define syntactic distance $d_{syn}(s_1, s_2)$ as:

$$d_{syn}(s_1, s_2) = \frac{d_{len}(s_1, s_2) + d_{char}(s_1, s_2) + d_{type}(s_1, s_2)}{3}$$

where d_{len} represents the difference in the number of tokens, d_{char} the difference at the character level, and d_{type} the difference based on certain predefined types (e.g., numeric values, email address, date and time, etc) as determined by regular expressions. The exact definition of these functions can be found in Appendix I.

Note that the use of syntactic features for segmentations has been extensively studied [3,8,16]. Our syntactic distance function uses well-known ideas from existing literature, and we do not claim to make new contributions in this respect.

2.4 An Optimization-based Formulation

Using distance functions, we can formally quantify the goodness of an extracted table as follows.

The objective function. Given a list $L = \{l_1, l_2, \ldots, l_n\}$ containing n unsegmented lines, the table we extract $T = \{t_1, t_2, \ldots, t_n\}$ should be such that each record t_i is a segmentation of l_i, and furthermore every record has the same number of columns m.

Let $t[k]$ be the k-th column of record t. Given a segmented table T with m columns, we can define our objective function, termed *sum of pairs (SP) distance*, as the sum of distance of every pair of cell values, $t_i[k]$ and $t_j[k]$, in the same column. Namely,

$$SP_m(T) = \sum_{1 \le k \le m} \sum_{1 \le i < j \le n} d(t_i[k], t_j[k]) \quad (2)$$

Since the summations are commutative, we can rewrite as

$$SP_m(T) = \sum_{1 \le i < j \le n} \sum_{1 \le k \le m} d(t_i[k], t_j[k]) \quad (3)$$

Let $d(t_i, t_j)$ be the distance between two records, defined as

$$d(t_i, t_j) = \sum_{1 \le k \le m} d(t_i[k], t_j[k]) \quad (4)$$

We can simplify Equation (3) to have

$$SP_m(T) = \sum_{1 \le i < j \le n} d(t_i, t_j) \quad (5)$$

Example 3. *We revisit our running example in Figure 3. By Equation* (4)*, the distance $d(t_1, t_2)$ between t_1 and t_2 is calculated as:*

$$d(t_1, t_2) = d(\textit{Los Angeles}, \textit{Toronto}) + d(\textit{California}, \textit{null}) + d(\textit{United States}, \textit{Canada})$$

$SP_3(T)$ can then be written as the sum of all pairs distance, where $d(t_1, t_3)$ and $d(t_2, t_3)$ are defined similarly.

$$SP_3(T) = d(t_1, t_2) + d(t_1, t_3) + d(t_2, t_3)$$

The use of all pair distance as the objective is natural – it captures the idea that a table is coherent if all pairs of values in the same column are coherent. We note that sum of pairs is also used in clustering [5], bioinformatics [22], and facility location problems [19], among other things.

Problem statement. Suppose we know the number of columns that lists need to be segmented into, we can define the problem ***Table Segmentation Given Column Number*** as follows.

Definition 2. ***Table segmentation given column number.*** *Given a list $L = \{l_1, l_2, \ldots, l_n\}$ and the desired number of columns m. The problem here is to find a table segmentation $T = \{t_1, t_2, \ldots, t_n\}$ that minimizes the sum of pair distance $SP_m(T)$ over all possible table segmentations.*

In the unsupervised setting where the number of columns m is not given, we can test all possible m's up to some upper limit (e.g., the maximum number of tokens in all lines), and pick the m that produces the best SP score *per column*.

Definition 3. ***Unsupervised table segmentation.*** *Given a list $L = \{l_1, l_2, \ldots, l_n\}$, the problem of unsupervised segmentation is to find a segmentation $T = \{t_1, t_2, \ldots, t_n\}$ that minimizes the per column objective score $\frac{SP_m(T)}{m}$ over all possible table segmentations and m.*

Note that we use the *per column* SP score as the objective function in the unsupervised setting, because we need to normalize the aggregate SP score for a fair comparison between segmentations with different number of columns m.

Example 4. *When the number of columns m is 2, the table T' in Figure 4 achieves the best $SP_2(T')$ for list L in Figure 2. Note that values in the first column of T' have high semantic distance values, because their co-occurrence in Web tables corpus are almost non-existent. This is an artifact of the $m = 2$ column constraint. If we specify $m = 3$, we will have the table T in Figure 3, where semantic distances are very low. In this case, $\frac{SP_3(T')}{3}$ will be lower than $\frac{SP_2(T)}{2}$, indicating that 3-column is the right segmentation of L.*

t_1	Los Angeles California	United States
t_2	Toronto	Canada
t_3	New York City New York	USA

Figure 4: Table T' with minimal $SP_2(T')$

Notice that for ***unsupervised table segmentation***, since we can try all numbers of columns m up to $|l_{max}|$, where $|l_{max}|$ can be the maximum number of tokens in any line, or some reasonable upper limit of columns we expect to see in a table (e.g., 20), for the rest of the paper we will only discuss ***table segmentation given column number*** for unsupervised table segmentation and treat m as a given input.

3. THE TEGRA ALGORITHM

While the SP objective function naturally captures the desirable property of a good table, we show that the general problem of unsupervised table segmentation is NP-hard using a reduction from multiple sequence alignment.

Theorem 1. *The decision version of unsupervised table segmentation is NP-hard.*

A proof of Theorem 1 can be found in Appendix B.

In light of the hardness, we propose the TEGRA algorithm that solves the segmentation problem with an approximation guarantee. In particular, in Section 3.1, we design a mechanism to decompose the objective function into more tractable components with quality guarantees. In Section 3.2, we further optimize the segmentation algorithm using ideas reminiscent of A* search. These together produce an efficient table segmentation algorithm.

3.1 A Conceptual Approximation Algorithm

In the following, we define the notion of *anchor* and *anchor distance* to decompose the SP distance.

Definition 4. *Let record $t_i \in T$ be an anchor record, the anchor distance of t_i, denoted as $AD_m(t_i, T)$, is the sum of distances between t_i, and all other records in T.*

$$AD_m(t_i, T) = \sum_{t_j \in T, j \ne i} d(t_i, t_j) \quad (6)$$

Using the definition of anchor distance and symmetry of distance d, SP distance is decomposed as the half of the sum of the anchor distance for all anchors. Namely,

$$SP_m(T) = \sum_{1 \le i < j \le n} d(t_i, t_j) = \frac{1}{2} \sum_{1 \le i \le n} AD_m(t_i, T) \quad (7)$$

We will omit subscript m in $SP_m(T), AD_m(t_i, T)$ when the context is clear.

Example 5. *Consider the example table T in Figure 3. For each anchor record we have*

$$AD_3(t_1, T) = d(t_1, t_2) + d(t_1, t_3)$$
$$AD_3(t_2, T) = d(t_2, t_1) + d(t_2, t_3)$$
$$AD_3(t_3, T) = d(t_3, t_1) + d(t_3, t_2)$$

Given that $SP_3(T) = d(t_1, t_2) + d(t_1, t_3) + d(t_2, t_3)$, we have $SP_3(T) = \frac{1}{2}(AD_3(t_1, T) + AD_3(t_2, T) + AD_3(t_3, T))$.

Intuitively, transforming SP to AD is to make the problem more tractable. SP considers n^2 pairs of interaction between record pairs in T, where a change of segmentation in one record has complex implications. In comparison, when computing AD, once the segmentation of anchor t_i is fixed, the best (minimal) $AD(t_i, T)$ over all possible T can be found by segmenting all other lines *independently* against t_i.

Formally, let $s(l_i)$ [5] be a segmentation function that maps a line l_i to all possible segmentations. Similarly $s(L)$ maps a list L to all possible tables. Given a segmented t_i, the best segmentation for each line l_j against t_i, denoted by t_j^{i*}, is the one that achieves the minimal distance with t_i.

$$t_j^{i*} = \arg\min_{t_j \in s(l_j)} d(t_i, t_j) \quad (8)$$

Notice that t_i and the segmentations of all other lines t_j^{i*} together induce a table, denoted as $R(t_i)$.

$$R(t_i) = \{t_j^{i*} | 1 \le j \le n, j \ne i\} \cup t_i \quad (9)$$

where $R(t_i)$ is the *table induction function* for anchor t_i.

By the definition of the anchor distance AD, we know $R(t_i)$ minimizes $AD(t_i, R(t_i))$ over all possible tables.

$$R(t_i) = \arg\min_{T \in s(L)} AD(t_i, T) \quad (10)$$

Equation (10) follows from the fact that in the definition of $AD(t_i, T)$ (Equation (6)), only distance scores between anchor t_i and other records count, distance between other records, $d(t_j, t_k), j \ne i \wedge k \ne i$, do not contribute to AD. Thus, each line can be independently optimized in Equation (8), yet the union of all lines in Equation (9) still minimizes AD anchored on t_i (Equation (10)).

Notice that finding table segmentations $R(t_i)$ through AD minimization is more tractable, because only one pair (t_i, l_j) needs to be considered at a time, as opposed to considering n lines simultaneously in SP optimization.

Interestingly, we show that the table segmentation found by minimizing $AD(t_i, R(t_i))$ over all possible anchor t_i is not bad compared to the global optimal $SP(T)$.

Define t_i^* to be the segmentation of l_i that minimizes $AD(t_i^*, R(t_i^*))$ over all possible $t_i \in s(l_i)$:

$$t_i^* = \arg\min_{t_i \in s(l_i)} AD(t_i, R(t_i)) \quad (11)$$

[5]The segmentation function $s()$ is defined with subscript m in Definition 1 but also used without subscript when context is clear.

Let c be the index that minimizes $AD(t_i^*, R(t_i^*))$ over all choices of i:

$$(t_c^*, R(t_c^*)) = \arg\min_{1 \le i \le n} AD(t_i^*, R(t_i^*)) \quad (12)$$

Let $T^* = \arg\min_{T \in s(L)} SP(T)$ be the global optimal segmentation with minimum $SP(T)$. In the following theorem, we can show that the SP distance, $SP(R(t_c^*))$, of the segmentation $R(t_c^*)$ induced by picking the best t_c^*, is not far away from the global optimal $SP(T^*)$.

Theorem 2. *Let $R(t_c^*)$ be the table segmentation with minimum AD distance over all possible choices of anchors as defined in Equation* (12)*. Let T^* be the global optimal segmentation with the minimum $SP(T^*)$. If d satisfies triangle inequality, then $SP(R(t_c^*)) \le 2SP(T^*)$*

The idea here to bound $SP(R(t_c^*))$ using $SP(T^\star)$ is to utilize triangle inequality, the fact that $R(t_c^*)$ minimizes AD over all possible anchor segmentation, and the relationship between AD and SP outlined in Equation (7). A proof of this theorem can be found in Appendix C. Using this result, we develop `TEGRA-naive` in Algorithm 1 that has 2-approximation.

Algorithm 1: `TEGRA-naive`

Input: List $L = \{l_1, l_2, \ldots, l_n\}$, number of columns m
Output: Segmented table $T = \{t_1, t_2, \ldots, t_n\}$

```
1 for each l_i ∈ L do
2     for each t_i ∈ s(l_i) do
3         AD(t_i, R(t_i)) = 0
4         for each l_j ∈ L, j ≠ i do
5             t_j^{i*} = arg min_{t_j ∈ s(l_j)} d(t_i, t_j)
6             AD(t_i, R(t_i)) += d(t_i, t_j^{i*})
7     t_i^* = arg min_{t_i ∈ s(l_i)} AD(t_i, R(t_i))
8 c = arg min_{1≤i≤n} AD(t_i^*, R(t_i^*))
9 Return R(t_c^*)
```

`TEGRA-naive` works as follows. It first picks each line $l_i \in L$ as an anchor. Then for each anchor segmentation $t_i \in s(l_i)$, it segments every other line l_j to t_j^{i*} so that it minimizes distance $d(t_i, t_j^{i*})$ over all possible $s(l_j)$ (Line 5). The anchor segmentation t_i^* that has the minimum AD distance is picked (Line 7). This induces a table segmentation $R(t_i^*)$ using a table induction function $R()$ that unions all line segmentations in Equation (8), (9). We return as result the table $R(t_c^*)$ induced by anchor t_c^* that has minimum AD distance over all anchor lines.

Example 6. *Consider list L in Figure 2:*

For line l_1, `TEGRA-naive` tries all possible segmentations t_1. For every t_1 such as the one in Figure 3, it will find the best t_2^{1*} out of all possible t_2 for l_2 such that $d(t_1, t_2^{1*})$ is minimal and the best t_3^{1*} out of all possible t_3's such that $d(t_1, t_3^{1*})$ is minimal. It will remember the t_1^* that results in the minimal $AD(t_1^*, T_1^*) = d(t_1^*, t_2^{1*}) + d(t_1^*, t_2^{1*})$, where $T_1^* = \{t_1^*, t_2^{1*}, t_3^{1*}\}$.

Similarly, T_2^* is derived for l_2, and T_3^* for l_3.

`TEGRA-naive` returns one of T_1^*, T_2^*, T_3^* whose $AD(t_i^*, T_i^*)$ is minimal among all $1 \le i \le 3$.

While the naive version of the algorithm is logically clear, each of the two steps at Line 2 and Line 5 requires enumerating all possible segmentations $t \in s(l)$ for a line l, which is prohibitively expensive with a large number of columns. In the next section, we will explain how these two key components can be optimized. We also note that even in this naive

algorithm, when assuming the number of desired columns is no more than a fixed constant (e.g., it is shown that over 95% Web tables have less than 10 columns [10]), the complexity of `TEGRA-naive` is already a polynomial of the input.

3.2 Optimization Strategies

As discussed above, there are two places in `TEGRA-naive` that require exhaustive enumeration of possible segmentations of a line l. In order to optimize this, we need to address two problems efficiently: (1) **Pruning Anchor Segmentation (PAS)**: How to prune away unpromising segmentations so that the t_i^* that minimizes $AD(t_i^*, R(t_i^*))$ can be found efficiently without looking at all possible segmentations (Line 2 and Line 7 of Algorithm 1). (2) **Segment a Line Given a Record (SLGR)**: Given a segmented t_i, how to segment another line l_j such that $d(t_i, t_j)$ is minimized (Line 5 of Algorithm 1). For technical reasons, we will discuss SLGR first and then PAS in the following.

3.2.1 Segment a Line Given a Record (SLGR)

We are given an unsegmented line l_j, and a segmented record t_i. The goal is to find $t_j^{i*} \in s(l_j)$ that minimizes $d(t_i, t_j^{i*})$. Recall that the distance of two records $d(t_i, t_j)$ is defined as the sum of distance of all m pairs of cells in the same column, $\sum_{1 \le k \le m} d(t_i[k], t_j[k])$.

We define a *partial alignment cost function* $M_{l_j,t_i}[p, w]$, that calculates the cost of aligning the first p columns of t_i, using the first w tokens of l_j.

Definition 5. *The partial alignment cost function, denoted as $M_{l_j,t_i}[p, w]$, with $0 \le p \le m$ and $0 \le w \le |l_j|$, is defined as follows.*

$$M_{l_j,t_i}[p, w] = \min_{t' \in s_p(l_j[1 \ldots w])} \sum_{1 \le k \le p} d(t'[k], t_i[k]) \qquad (13)$$

Recall that $l_j[1 \ldots w]$ represents the consecutive subsequence of l_j that ends at the w-th token, and $s_p(l_j[1 \ldots w])$ is the segmentation function that enumerates all possible p-column segmentations of $l_j[1 \ldots w]$.

When the context is clear we omit the subscript l_j, t_i in $M_{l_j,t_i}[p, w]$. Notice that by definition, the final segmentation corresponding to $M[m, |l_j|]$ is exactly the desired segmentation t_j^{i*}, which minimizes $d(t_i, t_j^{i*})$.

There is an *optimal sub-structure* [13] in $M[p, w]$, namely

$$M[p, w] = \min \begin{cases} M[p-1, x] + d(l_j[x+1 \ldots w], t_i[p]) \\ \qquad\qquad \forall 0 \le x < w \\ M[p-1, w] + d(\text{null}, t_i[p]) \end{cases} \qquad (14)$$

$M[p, w]$ is calculated based on where the first $p-1$ columns end. If the first $p-1$ columns end at a token $x < w$, then the p^{th} column is $l_j[x+1 \ldots w]$. Thus $M[p, w]$ equals $M[p-1, x]$ plus the distance of aligning $l_j[x+1 \ldots w]$ with $t_i[p]$. If the first $p-1$ columns end at token w, then the p^{th} column is null. Thus $M[p, w]$ equals $M[p-1, w]$ plus aligning null with $t_i[p]$. $M[p, w]$ takes the minimum of all those choices. This is clearly amenable to dynamic programming. The actual algorithm can be found in Appendix D.

Example 7. *Revisiting our running example, suppose t_1 in Figure 3 has been segmented. Given l_2 in Figure 2, in SLGR, we need to find the min-cost segmentation t_2^{1*} of l_2 so that $d(t_1, t_2^{1*})$ is minimized.*

Figure 5 shows a sample matrix $M[p, w]$. The final segmentation t_2^{1*} is also t_2 in Figure 3.

In the $M[p, w]$ matrix in Figure 5, $M[0, 1], M[0, 2]$ is initialized to be ∞ because the 0^{th} column (a hypothetic column) cannot consume any tokens. Suppose $d(null, *) = 0.9$. We have $M[1, 0], M[2, 0], M[3, 0]$ populated as $0.9, 1.8, 2.7$ respectively because of this.

Using Equation (14), it can be calculated that $M[1, 1] = d(l_2[1], t_1[1]) = d(\text{Toronto, Los Angeles}) = 0.3$. Similarly,

$$M[2, 1] = \min \begin{cases} M[1, 0] + d(\text{Toronto}, t_1[2]) = 0.9 + 0.5 = 1.4 \\ M[1, 1] + d(null, t_1[2]) = 0.3 + 0.9 = 1.2 \end{cases}$$

Thus $M[2, 1]$ has an arrow from $M[1, 1]$. The final result t_2^{1*} is constructed by back tracing the computation from $M[3, 2]$ via arrows. For instance, $t_2^{1*}[3]$ is "Canada" because $M[3, 2]$ comes from $M[2, 1]$, meaning the second token "Canada" is used in the third column.

p \ w	0	1 (Toronto)	2 (Canada)
0	0	∞	∞
1	0.9	0.3	0.8
2	1.8	1.2	1.1
3	2.7	2.1	1.5

Figure 5: Matrix $M[p, w]$ for aligning l_2 with t_1.

3.2.2 Pruning Anchor Segmentations (PAS)

In this section, we develop techniques to prune away unpromising anchor segmentations in $s(l_i)$ while still finding $t_i^* = \arg\min_{t_i \in s(l_i)} AD(t_i, R(t_i))$ correctly (Line 2 to Line 7 in Algorithm 1).

Example 8. *The following segmentation t_1', for instance, is a bad choice for segmenting l_1 in Figure 2.*

t_1'	*Los*	*Angeles California United*	*States*

This is because there is no subsequence of tokens in any other line of Figure 2, that can possibly have a low distance score with $t_1'[1]$ ("Los"). The same is true for the second column $t_1'[2]$ and third column $t_1'[3]$.

Conceptually, we need a way to systematically avoid such segmentations. In the following, we will first show that the space of possible segmentations $s(l_i)$ can be naturally encoded in a graph. We will then show that finding the best segmentation translates to finding a path on the graph with minimum cost, which motivates us to use ideas from A* for efficient pruning of segmentations.

Anchor Segmentation Graph. Efficient pruning of segmentations requires us to first represent the space of segmentations $s(l_i)$. We define a search graph G_i for l_i that can compactly encode all segmentations as follows. Nodes in the graph are labeled as $l_i[p, w]$ (abbreviated to $[p, w]$ when context is clear), for all $1 \le p \le m-1$, and $0 \le w \le |l_i|$. There is a special start node $[0, 0]$ and a target node $[m, |l_i|]$. There is an edge going from $[p_j, w_j]$ to $[p_k, w_k]$, if $p_k = p_j + 1$ and $w_j \le w_k$.

Notice that in this graph, a path from the start node to the end node represents a segmentation of l_i. In fact the union of all paths in the graph G_i captures all possible segmentations $s(l_i)$. This gives us a natural and compact representation of the search space.

More formally, a *complete path* from the start node to target node specifies a complete segmentation t_i for a line l_i. A *partial prefix path* is a path from the start node to $[p, w]$, specifying only the partial segmentations of the first p columns using the first w tokens. Similarly, a *partial suffix*

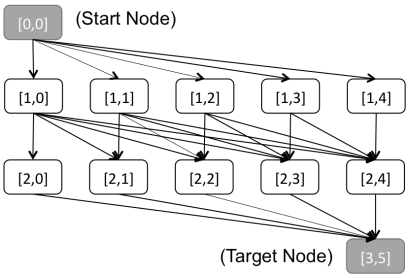


Figure 6: Search space G_1 for segmenting l_1

path is a path from $[p, w]$ to the target node, specifying only the partial segmentations of the last $m - p$ columns using remaining tokens starting at position $w + 1$.

Example 9. *Figure 6 shows the search graph for segmenting l_1 into three columns. The start node is $[0, 0]$. The is target node $[3, 5]$, because l_1 has five tokens and we need to segment into three columns.*

The complete path $Z : [0, 0] \rightarrow [1, 2] \rightarrow [2, 3] \rightarrow [3, 5]$ fully specifies one possible segmentation: the first column is "Los Angeles" (using 2 token for 1 column), the second column "California" (3 token for 2 columns), and the third column "United States" (5 tokens for 3 columns).

The partial prefix path $X : [0, 0] \rightarrow [1, 2]$ corresponds to a partial segmentation, where the first column is "Los Angeles", and the last two columns are yet to be segmented using the remaining three tokens "California United States".

Similarly the partial suffix path $Y : [2, 3] \rightarrow [3, 5]$ corresponds to a partial segmentation with only the third column specified as "United States".

Having created G_i, we need to assign cost/lengths to graph paths, to represent the current distance cost of the (partial) segmentation.

Definition 6. *Path length in G_i is defined as follows.*

• ***Length of a complete path Z.*** *Let t_i^Z be the segmentation specified by path Z. The length of the path $L(Z)$ is:*

$$L(Z) = \sum_{\substack{1 \le j \le n \\ j \ne i}} M_{l_j, t_i^Z}[m, |l_j|]$$

Where M_{l_j,t_i} is the alignment cost function defined in Definition 5. Note that here all m columns and $|l_j|$ tokens have been aligned.

• ***Length of a partial prefix path X to node $[p, w]$.*** *Let t_i^X be the partial segmentation specified by X. The length of the path $L(X)$ is defined as:*

$$L(X) = \sum_{\substack{1 \le j \le n \\ j \ne i}} \min_{0 \le k \le |l_j|} M_{l_j, t_i^X}[p, k]$$

• ***Length of a partial suffix path Y from node $[p, w]$.*** *Let t_i^Y be the partial segmentation specified by Y. The length of the path $L(Y)$ is defined as:*

$$L(Y) = \sum_{\substack{1 \le j \le n \\ j \ne i}} \min_{\forall 0 \le k \le |l_j|} N_{l_j, t_i^Y}[p, k]$$

where $N_{l_j,t_i^Y}[p, w]$ is defined similar to M only backwards: it represents the minimal sum of distance between the last $m - p$ columns of t_i, and the last $m - p$ columns starting at the $(w + 1)$-th token of l_j. Matrix N can be calculated similar to M using Algorithm 3.

Note that the length of a complete path Z is the sum of distances between t_i^Z and all other lines, which is exactly our anchor distance $AD(t_i^Z, R(t_i^Z))$.

t_1^X	Los Angeles	California United States
t_2^X	Toronto	Canada
t_3^X	New York	City New York USA

Figure 7: Calculating the length of partial prefix path X. The gray cells are unsegmented.

When calculating the length of a partial prefix path X, since the remaining $m - p$ columns of t_i are not fully segmented yet, we do not know the best way to segment the first p columns in l_j to align with t_i. Therefore, we test all possible ending tokens for the first p columns in l_j to align with $t_i[1 \dots p]$. The length of a partial suffix path $L(Y)$ is defined similar to $L(X)$ except that the backward DP matrix N is used.

Our definition of path lengths is not additive and thus different from traditional graph paths. Namely, for some partial prefix path X ending at $[p, w]$, partial suffix path Y starting from $[p, w]$, and complete path $Z = X \cup Y$, we have $L(X) + L(Y) \neq L(Z)$. However, we show that our paths are *super-additive*, which is a nice property.

Lemma 1. *The paths of search graph G_i defined above are super-additive. That is, for any partial prefix path X ending at $[p, w]$, partial suffix path Y starting from $[p, w]$, and complete path $Z = X \cup Y$, we have $L(X) + L(Y) \le L(Z)$.*

Intuitively, $L(X) + L(Y) \le L(Z)$ is true because the definition of partial paths allows other lines l_j to use a flexible number of tokens to align with p columns from t_i, thus giving partial paths more freedom in minimizing their lengths, making the sum of lengths of two partial paths smaller than the length of the full path. A proof of Lemma 1 can be found in Appendix F. Super-additivity allows us to estimate (underestimate) $L(Z)$ by estimating $L(X), L(Y)$ separately.

Example 10. *For a partial prefix path $X : [0, 0] \rightarrow [1, 2]$ in G_1, the corresponding partial segmentation $t_1^X[1]$ is shown in Figure 7. Figure 7 also shows the partial segmentations of other lines $t_2^X[1], t_3^X[1]$, that have the minimal distance with $t_1^X[1]$ respectively. Notice that although in the final result "New York City" in t_3 should align with "Los Angeles" from t_1, here "New York" is aligned with "Los Angeles" when calculating $L(X)$ because it has a smaller distance with "Los Angeles" than "New York City".*

Pruning anchor segmentations (PAS) using $A^\star$ Search. Now that we have defined graph G_i to encode all segmentations of l_i, finding the minimum AD cost segmentation naturally translates to finding the shortest path, where ideas from A* search [23] are directly applicable.

In A* search, there is a concept called heuristic/future function h [23] , which gives an estimate of the cost of the path from an intermediate node to the target node (note that in spite of the name heuristic, it always finds the correct min-cost path). It can be shown that if h never overestimates the true cost of reaching the target, then A* can be used to safely prune away unpromising paths. We refer readers to Appendix E for a more detailed overview of A*.

Given the defined lengths of paths, it is important to design an appropriate h function that closely approximates but always underestimates the future cost. Specifically, the h function takes as input a node $[p, w]$, and returns an underestimate of the min-cost path from $[p, w]$ to the target.

Notice that while the past cost of a partial prefix path X at $[p, w]$ can be calculated following Definition 6, the future cost starting from $[p, w]$ cannot be calculated in the same way, because the future path Y is not known at this point yet. The idea then is to segment the remaining $m - p$ columns

in l_i while allowing other lines to align with these $m-p$ columns fully flexibly to produce a safe underestimate.

In order to achieve this, for any candidate column c in l_i that is $c = l_i[a \dots b]$, define a *free distance* of c as follows.

Definition 7. *The free distance of c and l_j, denoted as $freeD(c, l_j)$, is*

$$freeD(c, l_j) = \min_{1 \le a_j \le b_j \le |l_j|} d(l_i[a \dots b], l_j[a_j \dots b_j])$$

The free distance of c, denoted as $freeD(c)$, is simply the summation of $freeD(c, l_j)$ for all $l_j, j \neq i$.

$$freeD(c) = \sum_{\substack{1 \le j \le n \\ j \neq i}} freeD(c, l_j)$$

Intuitively, the free distance of c and l_j is the minimum distance between c and any subsequence in l_j regardless of other alignments (hence the name free).

Example 11. *In Figure 7, suppose we are calculating $freeD$("California") in t_1^X . Candidate column "Canada" in l_2, "New York" in l_3 have the smallest distance with "California" respectively. Notice that although in final table, "null" in l_2 is aligned with "California", in $freeD$ calculation "Canada" is used for "California" because they have a smaller distance.*

Our h function uses free distance to (under-)estimate future costs. In particular, Let $r = l_i[w+1 \dots |l_i|]$ be remaining tokens at $[p, w]$ for conciseness. Using $freeD$, we can define a heuristic function $h(p, w)$ as follows.

$$h(p, w) = \min_{t' \in s_{m-p}(r)} \sum_{1 \le k \le m-p} freeD(t'[k]) \qquad (15)$$

The heuristic function $h(p, w)$ is the minimal sum of free distances of all $(m-p)$-column segmentation of r. Notice that Equation (15) is similar to Equation (13). Thus they share a similar optimal sub-structure as in Equation (14). A similar DP can thus be used to compute $h(p, w)$.

We empirically observe that the heuristic function designed here is effective in pruning away unlikely paths and achieves significant efficiency improvements as demonstrated in the experiments.

Lemma 2. *$h(p, w)$ is admissible [23], i.e., it underestimates the length of any path from $[p, w]$ to target node. Furthermore, $h(p, w)$ is monotonic [23], i.e., $L(X) + h(p, w) \le L(X') + h(p+1, w')$ where X is a path from starting node to node $[p, w]$, and X' is a path extending X with an additional node $[p+1, w']$.*

The actual algorithm for calculating $h(p, w)$ and a proof of Lemma 2 can be found in Appendix G.

Algorithm 2 describes the $A^\star$ search procedure for finding T_i^* with the minimal $AD(t_i^*, T_i^*)$. It keeps a closed set of nodes, representing a set of visited node whose shortest path from start node has already been found, and a set of open nodes, representing a set of node to be visited ordered according to a node's f function. Each node $[p, w]$ in G_i stores the following information: (1) the current path X from start node to $[p, w]$ that has the minimal length, denoted as $[p, w].X$; (2) the length $L(X)$ of the current path X, i.e., the g function in $A^\star$ search; (3) the underestimated distance between the current node to the destination node, i.e., h function in $A^\star$ search; and (4) the f function in $A^\star$ search, which is equal to $g + h$. At each step of Algorithm 2, the node with the lowest f is removed from the set of open nodes, and the f, g, X of its neighbors are updated accordingly (Lines 8-12). The loop terminates when the target node is removed from open nodes (Line 5).

Algorithm 2: `Minimizing Anchor Distance`

Input: List L, Number of columns m, anchor line l_i
Output: $t_i^*, R(t_i^*)$, such that $AD_m(t_i^*, R(t_i^*))$ is minimized

```
1  Initialize the openset, add start node [0,0] to it
2  Initialize the closedset, initially empty
3  while openset ≠ ∅ do
4      [p,w] ← the node with the lowest f score
5      break if [p,w] is the target node, construct t_i^*
6      remove [p,w] from openset, add [p,w] to closedset
7      for neighbor nodes [p+1,w'] of [p,w] do
8          continue if [p+1,w'] is in the closedset
9          X' ← [p,w].X ∪ [p+1,w']
10         if [p+1,w'] ∉ openset or L(X') < g(p+1,w') then
11             update X, g, f of [p+1,w']
12             add [p+1,w'] to openset if it is not in there
13 construct R(t_i^*) by aligning every l_j with t_i^*
14 Return t_i^* and R(t_i^*)
```

Theorem 3. *Algorithm 2 terminates with a table T_i that has the minimal $AD(t_i^*, R(t_i^*))$.*

The correctness of Algorithm 2 follows directly from the proof of the correctness of $A^\star$ search [23] and the fact that our h function is admissible and monotonic as shown in Lemma 2.

We note that an improved algorithm `TEGRA` thus replaces Lines 2- 7 in `TEGRA-naive` with Algorithm 2.

4. THE SUPERVISED VARIANT

While we set out to design algorithms that can extract tables fully automatically, we note that our approach applies directly in the scenario where user supervision is provided.

In an online ad-hoc table extraction scenario (such as in spreadsheets), users may be willing to provide example segmentations for a few lines. Suppose in addition to the unsegmented lines $L = \{l_i | i \in [n]\}$, we have a set of user provided segmentation examples, denoted as $E = \{l_i | i \in [k]\}$, for some $k < n$.

In such cases, we can modify the objective function to have $SP'_m(T) = \sum_{1 \le i < j \le n} w_{ij} d(t_i, t_j)$, where w_{ij} reflects the relative importance of examples. This way, $SP'_m(T)$ can then assign more weights to record pairs involving user examples because they are generally more reliable.

Definition 8. ***Supervised table segmentation.*** *Given $L = \{l_i | i \in [n]\}$ and user segmented examples $E = \{l_i | i \in [k]\}$ for some $k < n$. Our goal is to produce $T = \{t_i | i \in [n]\}$ with m columns, such that $SP_m(T)$ is minimized.*

We present one way to assign w_{ij} given k user examples:

$$w_{ij} = \begin{cases} \frac{n}{k} & \text{if } t_i \in E \text{ or } t_j \in E \\ 1.0, & \text{otherwise} \end{cases}$$

These w_{ij} ensures that the user-provided examples are more important, i.e., $w_{ij} \ge 1$ if t_i or t_j is user provided. Furthermore, the importance of user examples $\frac{n}{k}$ are adjusted based on the total number of lines and number of user examples, so that the impact of user examples is neither overwhelming nor underwhelming.

The TEGRA approach applies directly in this supervised problem variant, which is also shown to outperform alternative approaches in our experiments.

5. EXPERIMENTS

We conduct extensive experiments to understand the performance of different algorithms.

Data set	avg # of rows	avg # of cols	avg % of numeric cells
Web	14.2	6.2	43.1%
Wiki	11.8	5.0	42.1%
Enterprise	15.0	4.5	56.8%

Table 1: Some characteristics of three data sets. Numbers are averaged over 10,000 tables.

5.1 Experimental Setup

5.1.1 Algorithms compared

• ListExtract [16]. This work addresses the exact same problem as ours. As discussed in Appendix A, ListExtract makes local splitting decisions first, and then tries to make adjustment to the local decisions. It is the most relevant work directly comparable to TEGRA.

• Judie [14]. Judie represents a large class of record segmentation techniques [3, 14, 28] that rely on the existence of a reference table / knowledge base (KB) in the matching domain of the input list (e.g., a recipe knowledge base [14]). While it has been shown to work well when the matching KB in that narrow domain is provided, it is unclear how this class of approaches perform when segmenting general lists on the Web using general purpose KBs. We compare with a recent work Judie that is shown to outperform other techniques, including the unsupervised CRF U-CRF [28]. We use the popular Freebase [6] as the general purpose KB.

• TEGRA. This is our method discussed in Section 3.

5.1.2 Data preparation

We crawl two table corpora for our experiments.

First, we crawl over 100 million tables from the document index of a commercial search engine, henceforth referred to as Web-All. These tables are classified as high-quality relational content (e.g., not HTML tables for formatting, etc.). This corpus covers diverse relational content in almost all possible domains available on the public Web.

We also crawl over 500,000 tables from Excel spreadsheets in the Intra-net of a large IT company. This is referred to as Enterprise-All.

5.1.3 Benchmark table sets

In order to evaluate quality of tables extracted, for each unsegmented list we need a ground truth table that is the correct segmentation of the list. Manual labeling cannot scale to a test set large enough to draw any reliable conclusion, due to the high cost of labeling (authors in [16] for instance manually labeled segmentations for 20 lists). In order to perform large scale experiments for a reliable quality comparison, yet without actually labeling data, we automatically construct test datasets similar to [16]. We randomly sample existing tables, and then we concatenate cells in the same row into a line using white-spaces as delimiters. For instance, the table in Figure 3 crawled from the Web is concatenated into a list in Figure 2 and used as input. The original table can then be used as the ground truth to evaluate segmentation quality.

We prepare three datasets of different characteristics as our benchmark sets, namely Web, Wiki and Enterprise, each with 10,000 tables. The Web dataset is randomly sampled from Web-All, which contains relational tables crawled from the Web. The Wiki dataset is also sampled from Web-All but restricted to the *wikipedia.org* domain, which are generally of high quality compared to the Web dataset. The Enterprise benchmark set is sampled from Enterprise-All that contains spreadsheets crawled from an enterprise. Table 1 reports some characteristics of these three data sets.

As an additional sanity check to validate the large scale evaluation using automatic methods above, we also manually sample 20 real Web lists spanning across domains such as airports, movies, cartoons, presidents, and sports. We manually segment these lists into tables as ground truth. Note that these lists use many different delimiters such as comma, semicolon, dash, and tab. This dataset is henceforth referred to as Lists.

5.1.4 The background table corpus

Given that our semantic coherence requires co-occurrence computation like PMI from a background table corpus to determine semantic distance (Section 2.3.1), we use Web-All and Enterprise-All for this purpose.

The Web-All corpus covers diverse relational content in almost all possible domains available on the public Web. We construct the background corpus Background-Web as Background-Web = Web-All \ (Web ∪ Wiki).

Note that we intentionally exclude benchmark sets from the background corpus, to ensure that the quality numbers genuinely reflect algorithms' capability of segmenting tables not seen before. This is similar to the classical training set / testing set separation used in Machine Learning.

Given the excellent coverage of Background-Web, and the fact that it is of public nature such that anyone with sufficient resources could crawl this corpus, we use this as the default background corpus across our experiments.

In addition, we create a background corpus Background-Enterprise = Enterprise-All \ Enterprise. Since the benchmark set Enterprise is obtained from the same source as Background-Enterprise, Background-Enterprise provides additional co-occurrence information specific to the Enterprise test set (e.g., organization names, customer names, etc.). Thus, we report quality of extracting Enterprise using Background-Enterprise in addition to Background-Web.

5.1.5 Evaluation metric

Direct comparisons between an output table T_a and the ground truth table T_g can be tricky, since there may be multiple, equally correct ways of segmenting lists into tables, and T_a and T_g may have different number of columns. For instance, suppose Table 2 is the ground truth table T_g we crawled, and Table 3 is the output T_a of an algorithm. Although first-name/last-name are separated into two columns in ground truth T_g, they are concatenated in T_a. Similarly month/date columns are segmented in T_a but not T_g.

Table 2: T_g

Jenny	Scott	Jan 12
John	Smith	Nov 20

Table 3: T_a

Jenny Scott	Jan	12
John Smith	Nov	20

Intuitively, both segmentations are reasonable in this particular case, since we could easily find tables on the Web that segment first-name/last-name and month/date in two columns, as well as tables that concatenate them. Even human labelers may disagree on the right segmentation.

Our observation is that segmenting one column into multiple ones, or conversely concatenating multiple consecutive columns into one still provides some utility. For instance, regardless of whether first-name/last-name are concatenated or not, the extracted table still provides useful information. However, traditional set-based precision/recall evaluation does not take this into account.

We are not the first to recognize this problem. It has been shown [15, 18] that in certain matching and alignment problems, traditional precision/recall is not suitable, and generalized P/R evaluations has been proposed [15, 18]. In

the following we describe the metric that we use, which is similar in spirit to [18].

Define a *column mapping* M as a mapping from one column in T_g to multiple consecutive columns in T_a, or one column in T_a to multiple consecutive columns in T_g. Denote by $M(T)$ the column(s) of T that participates in M. Also let $l(M(T))$ be the values of columns $M(T)$ in line l.

Define $|M|$ as the total number of correctly aligned row values between columns $M(T_a)$ and $M(T_g)$, namely $|M| = |\{i : l_i(M(T_a)) = l_i(M(T_g)), i \in [n]\}|$.

A *mapping set* $\mathcal{M} = \{M_i : i \in [k]\}$ is a set of column mappings M_i such that no two different mappings touch overlapping columns in T_g and T_a. In other words, $\forall i \neq j$, $M_i(T_a) \cap M_j(T_a) = \emptyset$ and $M_i(T_g) \cap M_j(T_g) = \emptyset$.

Let $|\mathcal{M}| = \sum_{i \in [k]} |M_i|$, which is the number of correctly aligned values, and $|\mathcal{M}_{best}| = \max_{\mathcal{M}} |\mathcal{M}|$ be the best possible set of mappings.

We define the precision (P), recall (R), and f-measure (F) as follows:

$$P = \frac{|\mathcal{M}_{best}|}{|T_a|} \quad R = \frac{|\mathcal{M}_{best}|}{|T_g|} \quad F = \frac{2PR}{P+R}$$

where $|T|$ denotes the total number of cells in T.

As a concrete example, for T_g and T_a in Table 2 and 3, $\mathcal{M}_{best}$ has two column mappings: M_1 that maps the first two columns in T_g to the first column in T_a, where $|M_1| = 2$ (mappings in both rows are correct); and M_2 that maps the third column in T_g to the last two columns in T_a, where $|M_2| = 2$. $|\mathcal{M}_{best}| = |M_1| + |M_2| = 4$. $P = \frac{4}{6} = 0.67$ and $R = \frac{4}{6} = 0.67$. Note that using this metric, we only get partial credits when we over-segment or under-segment in a consistent manner. Observe that in this case where the produced a table T_a that is clearly useful, the our P/R values are only 0.67.

5.2 Overall table quality comparison

Unsupervised. Table 4 shows the the overall quality comparison when extracting tables from lists without any user input. It is clear from the table that TEGRA significantly outperforms both `ListExtract` and `Judie` across all benchmark sets. Notice that the difference in F-measures between TEGRA and `ListExtract` (the second-best method) is quite significant – the results are 0.9 vs 0.76, 0.9 vs 0.73 and 0.87 vs 0.79 for the three automatically generated test sets respectively, and 0.91 vs 0.7 for the manually label test set. This validates our belief that a principled formulation that performs global optimization is superior to `ListExtract` that uses local decisions that can be sub-optimal.

We observe that while the recall numbers of `ListExtract` come relatively close to TEGRA, its precision numbers lag much behind (0.89 vs 0.69, 0.89 vs 0.65, 0.88 vs 0.75, and 0.89 vs 0.63 respectively). We believe this is because of the way `ListExtract` works. Recall that it greedily extract the most confident cells from each row independently, which tends to lead to over-segmentation, because shorter token subsequences are naturally more popular in a language model and other sources (e.g., "New York" is a popular name and is likely to be extracted first despite the token "City" that comes after "New York"). Since it is more aggressive in splitting, its precision numbers suffer the most.

The `Judie` approach does not perform as well as others. This is not entirely surprising – the algorithm relies on a domain-specific reference table / knowledge base (KB) to perform segmentation correctly through essentially KB lookups. In the case of general lists on the web, even a general-purpose KB like Freebase [6] does not have nearly enough entities to cover values that can possibly occur in wide range of tables on the Web, not to mention various naming variations (e.g., a KB may have the entry "United Kingdom" but not "UK" or "Britain") and ambiguity of names (e.g., "United" may refer to airline or country). Thus, the class of techniques represented by `Judie` is not suitable for general table extraction from lists on the Web.

		TEGRA	ListExtract	Judie
Web	P	**0.89**	0.69	0.37
	R	**0.94**	0.88	0.45
	F	**0.90**	0.76	0.40
Wiki	P	**0.89**	0.65	0.44
	R	**0.94**	0.85	0.50
	F	**0.90**	0.73	0.46
Enterprise	P	**0.88**	0.75	0.43
	R	**0.89**	0.87	0.49
	F	**0.87**	0.79	0.45
Lists	P	**0.89**	0.63	0.60
	R	**0.94**	0.82	0.64
	F	**0.91**	0.70	0.62

Table 4: Quality Comparison (unsupervised)

		TEGRA	ListExtract	Judie
Web	$P/R/F$	**0.97**	0.70	0.39
Wikipedia	$P/R/F$	**0.96**	0.71	0.50
Enterprise	$P/R/F$	**0.94**	0.81	0.49
Lists	$P/R/F$	**0.95**	0.71	0.61

Table 5: Quality Comparison (supervised)

Supervised. We show an overall quality comparison when user feedback is given (i.e., the supervised setting) in Table 5. We emulate the user input by randomly selecting two records from the ground truth table as input.

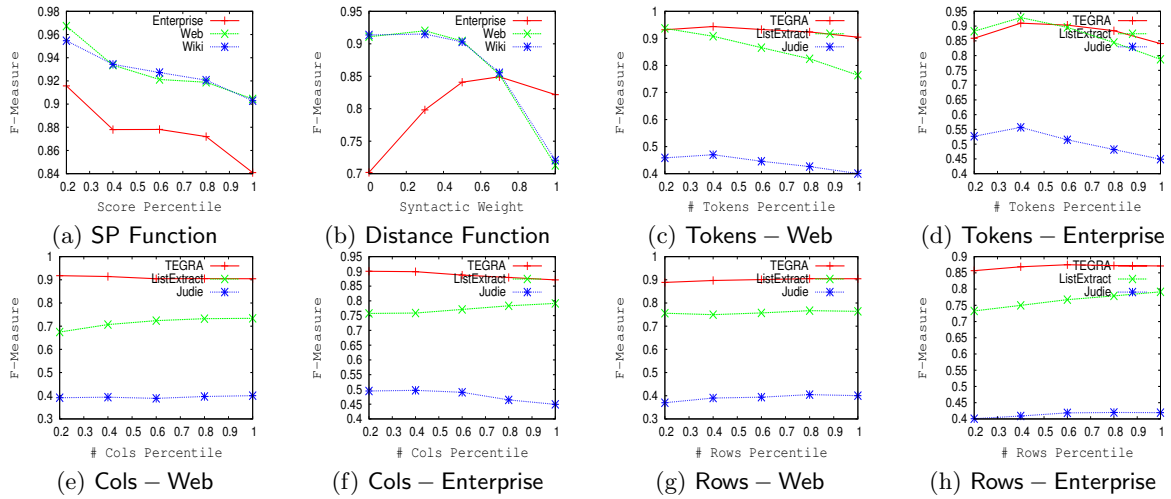
It is worth noting that while user feedback gives a quality boost to all three algorithms, TEGRA benefits the most. This demonstrates the usefulness of the proposed framework – using the same technique and a little user feedback, the quality of TEGRA can be greatly improved. We report additional experiments in the supervised setting in Appendix K due to space limit.

SP Distance Analysis. In order to analyze the correlation between our objective function and the "goodness" of tables extracted, we sort extracted tables based on their objective scores normalized on a per tuple-pair basis, and bucketize the tables into five bins. We plot the score bucket percentile on the x-axis and F-measure on the y-axis in Figure 8(a). The overall trend is that as the normalized objective score increases, table quality as measured by F-measure decreases across all three datasets. This validates the usefulness of our formulation and scoring – by minimizing SP distance we are indeed producing high quality tables.

5.3 Sensitivity to different tables

In this section we vary input tables to understand the performance of the three algorithms in response to tables of different characteristics. We skip the results on `Wiki` dataset since the results are similar to `Web`.

First, we observe that tables with more number of tokens per cell are generally more difficult to segment (because algorithms are more likely to make mistakes in picking wrong splitting positions). In contrast tables with fewer number of tokens per cell are easier, which are often tables dominated by 1-token numerical columns that are easy to seg-



(a) SP Function (b) Distance Function (c) Tokens – Web (d) Tokens – Enterprise

(e) Cols – Web (f) Cols – Enterprise (g) Rows – Web (h) Rows – Enterprise

Figure 8: Experiments in the unsupervised setting

ment. Using this observation, we sort tables based on their average number of tokens per cell, which is a proxy measure of segmentation "difficulty". We again bucketize tables into 5 buckets, and plot the corresponding F-measure in Figures 8(c) and 8(d).

When the average number of tokens per cell is low, the F-measure of `ListExtract` and `TEGRA` are very close, since a large fraction of these tables are actually dominated by 1-token numerical columns, which are not difficult to segment. As the average number of tokens per cell increases, segmentation becomes non-trivial, and the F-measure drops quite significantly for `ListExtract`. Note that even in the hard-to-segment cases, `TEGRA` performs surprisingly well – in fact, its quality numbers stay almost the same across all five table buckets. The fact that `TEGRA` can segment difficult (and more meaningful) tables almost as well as trivial 1-column numerical tables underlines the robustness of TEGRA.

Second, we vary tables by the number of columns. We observe in Figures 8(e) and 8(f) that as the number of columns increases, quality results only drop slightly. This is somewhat surprising and contrary to our initial belief that tables with more columns are more difficult to segment. An analysis reveals that tables with more columns are actually mixed, with both tables that are genuinely difficult to segment, and tables with mostly 1-token numerical columns that are trivial to segment. Since our test sets are uniformly sampled from the corpus, in aggregate the results are not sensitive to the number of columns.

Finally, we bucketize tables by the number of rows. We observe in Figures 8(g) and 8(h), as the number of rows increases, quality results of all three algorithms stay relatively the same. This is mostly expected – the number of rows does not correlate much with the hardness of segmentation.

5.4 Sensitivity to distance scoring

In this section, we vary the relative importance of syntactic distance and semantic distance, namely α in Equation (1), with the goal of understanding the importance of score components in different settings. In Figure 8(b), we vary the weight of the syntactic component α used in the distance function in Equation (1).

For `Web` and `Wiki`, starting with only semantic distance ($\alpha = 0$) the quality of `Web` and `Wiki` is already fairly good. As we increase the importance of syntactic distance the quality results improves slightly. When we move to the rightmost point with only syntactic distance ($\alpha = 1$), quality numbers suffer. This shows the importance of semantic distance for `Web` and `Wiki`, where syntactic features like the number of tokens is not sufficient to capture semantic coherence between values like "New York City" and "Toronto".

On the other hand, on the `Enterprise` data set, the quality results using only semantic distance ($\alpha = 0$) are not very good. As we increase the importance of syntactic distance quality numbers improve significantly. However when we reach the extreme with only syntactic distance ($\alpha = 1$) the performance falls again. This shows that syntactic distance is perhaps more important in the `Enterprise` data set, because a significant fraction of data is proprietary (e.g., customer names, internal ID codes, etc.) that cannot be found in `background-Web` crawled from the public domain. When semantic distance is not as indicative, syntactic distance naturally becomes more important.

Across all three benchmark sets `Web`, `Wiki` and `Enterprise`, both syntactic distance and semantic distance are shown to be important – using some combination of syntactic distance and semantic distance gives the best performance. This demonstrates the usefulness of both scoring components. Empirically we find $\alpha = 0.5$ to be a good setting (equal weights between semantic and syntactic distance), which is used as default in our experiments.

5.5 Impact of matching background corpus

In Table 6, we evaluate the effect of using different background corpora, namely, `background-Enterprise`, `background-Web` and `background-Combined` (which combines the previous two). It is quite clear that in all these cases, using the matching background corpus (e.g., `background-Enterprise` for test set `Enterprise`) or the combined corpus (`background-Combined`) provides the best performance.

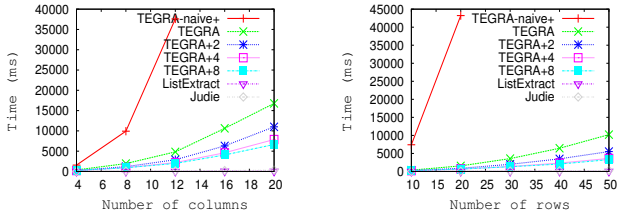
On the other hand, when using *mismatched* corpus, e.g., `background-Enterprise` for `Web`, the performance drops significantly. We believe that `background-Enterprise` does not really cover the diverse data found on the Web, thus cannot provide meaningful semantic distance. Surprisingly, using `background-Web` on `Enterprise` performs reasonably well, perhaps because `background-Web` is already diverse enough to cover certain aspect of `Enterprise` (e.g., geography data, product data, etc.). This is encouraging, because it show that the general purpose approach of using the public `background-Web` can actually provide reasonable performance on proprietary `Enterprise` data, which underscores the general applicability of our approach.

5.6 Efficiency study

Figures 9(a) and Figure 9(b) show the scalability of different approaches when varying the number of columns and number of rows, respectively. It is not really surprising that the `TEGRA` line (second from the top) takes considerably more

Test-Dataset \	Background	TEGRA	ListExtract	Judie
Web	B-Web	**0.90**	**0.76**	0.40
	B-Enterprise	0.68	0.73	0.40
	B-Combined	**0.90**	**0.76**	0.40
Wiki	B-Web	**0.90**	**0.73**	0.46
	B-Enterprise	0.70	0.71	0.46
	B-Combined	**0.90**	**0.73**	0.46
Enterprise	B-Web	0.84	0.78	0.45
	B-Enterprise	**0.87**	**0.80**	0.45
	B-Combined	**0.87**	0.79	0.45

Table 6: F-Measure using different background corpus (unsupervised)



(a) Varying # columns (b) Varying # rows

Figure 9: Scalability Analysis

time than ListExtract and Judie, since both use greedy procedures that lack formal quality guarantees. Given the significant quality improvement TEGRA provides, and the fact that our main targeted application is to extract tables from Web lists offline, we believe that the cost in execution time can be justified (we scale out the extraction process using a production Map Reduce system).

In comparison to TEGRA, TEGRA-naive+ line here uses the dynamic programming in SLGR, but no A* optimization is performed. TEGRA-naive+ actually takes over 40 seconds to segment a table with 20 rows, and over a few minutes when the number of rows increases (off the chart in Figure 9(b)). The significant improvement of TEGRA over TEGRA-naive+ shows the effectiveness of our A* optimization in Section 3.

We note that previous studies [16] as well as our own show that the majority (over 95%) of Web tables have less than 10 columns, in which case the unoptimized TEGRA can already finish within 4 seconds on average as shown in Figures 9(a). Furthermore, since the anchor-distance calculations are trivial to parallelize on a per row basis, we also experiment using multi-threading to see its impact on latency (denoted as TEGRA+n, where n is the number of threads used). We note that with modest multi-threading, e.g. TEGRA+4, we can reduce segmentation time for a 10-column table to around 1 second, which may already be acceptable for certain online applications like interactive table segmentation.

5.7 Estimate useful lists on the Web

As an empirical analysis of the number of useful relational tables in lists, we extract a sample of 770K Web HTML lists embedded in $\langle ul\rangle\langle/ul\rangle$ HTML tags from a small fraction of the Web crawl. We filter away lists with too many or two few rows, and those with very long rows, which leaves us with around 40K lists. We run our segmentation algorithms on these lists. The number of resulting tables that have a good empirical per-tuple quality score (based on Figure 8(a)) is around 2K. Given that the sampled Web chunk is about 0.006% of the whole index, we estimate that there are over 30 million lists on the Web with good relational content, which is in line with what is reported in [16].

6. RELATED WORK

The pioneering work in [16] is the only work that automatically extracts tables from general Web lists to our knowledge, and is reviewed in Appendix A. Because it greedily makes local splitting decisions for each line independently early in the process without giving consideration to global alignment, it can get stuck with sub-optimal local decisions.

There are several approaches, e.g., Judie [14], CRAM [3], U-CRF [28], that segment text into records utilizing a knowledge base (KB) or a reference table in the *matching* domain of the target content. Our experience shows that for these approaches to work well, a large fraction of values to be segmented must be covered in the matching KB. In the case of Web-scale table extraction where finding a matching KB for each target list is infeasible, using general KB like Freebase [6] does not work well (Section 5). We believe the reason lies in the fact that data on the Web are so diverse that even a general KB cannot provide sufficient coverage for the variety of values that can possibly occur in tables. In contrast, TEGRA obviates the need for specific KBs by taking advantage of statistical co-occurrence using a large table corpus that provides exhaustive content coverage.

Graphical models like CRF have been demonstrated to be useful for table segmentation, especially in supervised settings [21]. A recent approach, U-CRF [28], adapts CRF for the unsupervised setting and can be used without training examples. It is conceptually similar to Judie [14] in that it also requires matching KBs to be provided. Since it is shown in [14] that Judie outperforms U-CRF for unsupervised record parsing, we only compare with Judie in this paper. HMM has also been adapted for record segmentation in [8,25]. However, [8] uses training examples, and [25] requires explicit target schema, which make them unsuitable for extraction of tables from Web lists.

Our problem is also related to information extraction, and more specifically wrapper induction for structured Web data extraction [4, 12, 17]. However, all these approaches above are supervised in nature, requiring some examples to be given, which cannot scale to the general problem of extracting tables from millions of lists on the Web. Example driven extraction using program synthesis techniques (e.g., [20]) is another related line of work.

Sequence alignment in bioinformatics [22] addresses the problem of aligning biological sequences to identify similar regions, where similarity is predefined for pairs of symbols. The all-pairs objective function we use is reminiscent of the sum-of-pairs in the multiple sequence alignment (MSA) literature [22]. However, because symbols are always the unit of alignment in MSA, when pairs of sequences (a, b) and (a, c) are aligned, it is trivial to align the three sequences (a, b, c) together. In contrast, in table alignment this property does not hold, because token sequences become the unit of alignment and additivity of tokens no longer holds, which makes table alignment more difficult.

7. CONCLUSIONS AND FUTURE WORK

Motivated by the need to extract relational tables from HTML lists, we propose a principled formulation for the problem of table extraction. We design efficient algorithms with 2-approximation guarantees, which produce tables of superior quality compared to the state-of-the-art. Our approach also shows great promise in the supervised setting when user examples are provided, which is an interesting area that warrants further investigations.

8. REFERENCES

[1] Google Web Tables. `http://research.google.com/tables`.

[2] Microsoft Excel Power Query. `http://office.microsoft.com/powerbi`.

[3] E. Agichtein and V. Ganti. mining reference tables for automatic text segmentation. In *KDD*, 2004.

[4] E. Agichtein and L. Gravano. Snowball: extracting relations from large plain-text collections. In *DL*, 2000.

[5] Y. Bartal, M. Charikar, and D. Raz. Approximating min-sum clustering in metric spaces. In *STOC*, 2001.

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

[7] P. Bonizzoni and G. D. Vedova. The complexity of multiple sequence alignment with sp-score that is a metric. In *Theoretical Computer Science*, 2001.

[8] V. R. Borkar, K. Deshmukh, and S. Sarawagi. Automatic segmentation of text into structured records. In *SIGMOD*, 2001.

[9] M. J. Cafarella, A. Y. Halevy, and N. Khoussainova. Data integration for the relational web. In *VLDB*, 2009.

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

[11] K. W. Church and P. Hanks. Word association norms, mutual information, and lexicography. In *Computational Linguistics*, 1990.

[12] W. W. Cohen, M. Hurst, and L. S. Jensen. A flexible learning system for wrapping tables and lists in html documents. In *WWW*, 2002.

[13] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. 2001.

[14] E. Cortez, D. Oliveira, A. S. da Silva, E. S. de Moura, and A. H. F. Laender. Joint unsupervised structure discovery and information extraction. In *SIGMOD*, 2011.

[15] H. Do, S. Melnik, and E. Rahm. Comparison of schema matching evaluations. In *Workshop on Web Databases*, 2002.

[16] H. Elmeleegy, J. Madhavan, and A. Y. Halevy. Harvesting relational tables from lists on the web. *PVLDB*, 2009.

[17] D. W. Embley, Y. Jiang, and Y.-K. Ng. Record-boundary discovery in web documents. In *SIGMOD*, 2009.

[18] J. Euzenat. Semantic precision and recall for ontology alignment evaluation. IJCAI'07, 2007.

[19] R. L. Francis, T. J. Lowe, and H. D. Ratliff. Distance constraints for tree network multifacility location problems. In *Operations Research*, 1978.

[20] S. Gulwani, W. R. Harris, and R. Singh. Spreadsheet data manipulation using examples. In *CACM*, 2012.

[21] R. Gupta and S. Sarawagi. Answering table augmentation queries from unstructured lists on the web. In *PVLDB*, 2009.

[22] D. Gusfield. Efficient methods for multiple sequence alignment with guaranteed error bounds. *Bulletin of Mathematical Biology*, 1993.

[23] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *SIGART Bull.*, 1972.

[24] W. Just. Computational complexity of multiple sequence alignment with sp-score. In *Journal of Computational Biology*, 2001.

[25] A. Machanavajjhala, A. S. Iyer, P. Bohannon, and S. Merugu. Collective extraction from heterogeneous web lists. In *WSDM*, 2011.

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

[27] M. Zhang and K. Chakrabarti. Infogather+: Semantic matching and annotation of numeric and time-varying attributes in web tables. In *SIGMOD*, 2013.

[28] C. Zhao, J. Mahmud, and I. Ramakrishnan. Exploiting structured reference data for unsupervised text segmentation with conditional random fields. In *SDM*, 2008.

APPENDIX

A. ALGORITHM OVERVIEW IN [16]

Given an input list $L = \{l_1, l_2, \ldots, l_n\}$, `ListExtract` in [16] extracts a table $T = \{t_1, t_2, \ldots, t_n\}$ in three phases: an *independent splitting* phase, an *alignment* phase, and an *refinement* phase.

In the first phase, each line l_i is independently split into a multi-column record t_i based on *Field Quality Score* $FQ(f)$. $FQ(f)$ has three components: type support, language model support, and table corpus support.

In the alignment phase, since each line might be split into different number of columns in the previous step, the algorithm use the most common columns number m to re-split all lines. Records with number of columns less than m are expanded by inserting empty columns using a dynamic programming procedure to maximize consistency among cells in the same column. A *Field-to-Filed Consistency Score*, $F2FC(f_1, f_2)$, is used to measure the consistency of two fields f_1, f_2 being the same column. Records with number of columns more than m are merged and re-split into a record using the same algorithm in the independent splitting phase given m as a constraint.

In refinement phase, `ListExtract` tried to rectify some local decisions made before by identifying sequences of fields (termed streaks) in a record that are deemed to be incorrect. A field is deemed to be incorrect if its score with other fields in the same column is less than a predefined threshold. Each streak is then merged together and re-split.

B. PROOF OF THEOREM 1

We obtain the hardness result by a reduction from multiple sequence alignment (MSA) [24]. Recall that given a set $S = \{s_i | 1 \le i \le k\}$ of k strings over an alphabet Σ, a multiple alignment of is a $k \times l$ matrix A where row i contains string s_i interleaved by spaces. The score for a multiple alignment is the sum of all pairs of rows in the alignment, where the cost of a pair of rows is in turn the sum of all aligned symbol pairs. The penalty costs of all symbol pairs $P : (\Sigma, \Sigma) \to \mathbb{R}$ are pre-defined. The decision version of MSA is to determine if there exists a multiple alignment with cost no more than some given constant C.

For any instance of MSA, we construct an Unsupervised Table Segmentation(UTS) as follows. We create k unsegmented lines $L = \{s_i | 1 \le i \le k\}$ exactly using S, and define the cost function in UTS for each pair of token subsequences as:

$$d(l_i[a_i \ldots b_i], l_j[a_j \ldots b_j]) = \begin{cases} P(l_i[a_i \ldots b_i], l_j[a_j \ldots b_j]), \\ \qquad \text{if } a_i = b_i \wedge a_j = b_j \\ \infty, \text{ if } a_i \neq b_i \vee a_j \neq b_j \end{cases}$$

That is, if $l_i[a_i \ldots b_i]$ and $l_j[a_j \ldots b_j]$ are single symbols, then we use exactly P from MSA, otherwise we force it to ∞.

We show that if there exists a solution to MSA, i.e., a multiple alignment with cost at most C, then there is a solution to UTS with cost at most C. Suppose the multiple alignment solution A has l number of columns. It can be shown that $l \le \sum_{1 \le i \le k} |s_i|$. In the corresponding UTS, if we try all number of columns $m \le \sum_{1 < i < k} |s_i|$, then when using l number of columns, the best UTS will have a cost at most C. This is because the alignment A is a feasible

segmentation of L in UTS with exactly the same cost. Thus the solution to UTS must have a cost at most C.

We then show it in the other direction – if there exists a solution to UTS, i.e., a table segmentation with cost at most C, then there is a solution to MSA with cost at most C. Since there is a table segmentation with cost at most C, then all columns in the segmentations must be single-symbol or null (otherwise the cost is ∞ by construction). Thus this table segmentation is also a valid multiple alignment, meaning the solution to MSA must have a cost at most C.

Since MSA is shown to be NP-hard [24], this completes our hardness proof. Note that MSA is shown to be NP-hard even with metric distance [7]. It follows that UTS is also NP-hard even with metric distance.

C. PROOF OF THEOREM 2

Let $T_c^* = R(t_c^*)$ be the table returned by Algorithm 1 that has the minimal $AD(t_c^*, T_c^*)$, and $T^\star$ be the table with the minimal SP distance. Then we have

$$\begin{aligned} SP_m(T_c^*) &= \frac{1}{2} \sum_{1\le i\le n} \sum_{\substack{1\le j\le n \\ j\neq i}} d(t_i, t_j) \\ &\le \frac{1}{2} \sum_{1\le i\le n} \sum_{\substack{1\le j\le n \\ j\neq i}} \big(d(t_i, t_c^*) + d(t_c^*, t_j)\big) \qquad (16) \\ &= (n-1) \sum_{\substack{1\le i\le n \\ i\neq c}} d(t_c^*, t_i) \\ &\le (n-1) AD(t_c^*, T_c^*) \qquad (17) \end{aligned}$$

Note that Equation (16) holds by triangle inequality.

Overall, Equation (17) shows that $SP_m(T_c^*)$ can be bounded by $AD(t_c^*, T_c^*)$ from above. We further show that $SP_m(T^\star)$ can be bounded by $AD(t_c^*, T_c^*)$ from below.

$$\begin{aligned} SP_m(T^\star) &= \frac{1}{2} \sum_{1\le i\le n} AD(t_i, T^*) \quad \text{(By definition of AD)} \\ &\ge \frac{1}{2} \sum_{1\le i\le n} AD(t_i, T_i^*) \qquad (18) \\ &\ge \frac{n}{2} AD(t_c^*, T_c^*) \quad \text{(By definition of } t_c^*, T_c^*) \qquad (19) \end{aligned}$$

Note that Equation (18) is true because T_i^* is defined to be the table that minimizes anchor distance of t_i among all possible tables segmentations, which include T^*.

Dividing (17) by (19), we have:

$$\frac{SP_m(T_c^*)}{SP_m(T^\star)} \le \frac{2(n-1)nAD(t_c^*, T_c^*)}{nAD(t_c^*, T_c^*)} = 2 - \frac{2}{n} < 2$$

D. ALGORITHM FOR SLGR

Algorithm 3 describes a dynamic programming procedure to calculate $M[p, w]$ and the optimal segmentation t_j^{i*}. Lines 1-5 initialize $M[p, w]$ and address boundary cases. Line 8 uses the optimal substructure for calculating $M[p, w]$. The complexity of the procedure is $O(m|l_j|^2)$, where the size of the M matrix is $O(m|l_j|)$, and the calculation of each cell requires $O(|l_j|)$ comparisons.

E. $A^\star$ SEARCH OVERVIEW.

$A^\star$, an informed search algorithm, is designed to find a least-cost path from a start node to an end node quickly. It keeps a priority queue of alternate paths sorted by their underestimated cost to the end node. It traverses the graph by following a path that has least expected cost.

Algorithm 3: `Segment a Line Given Record (SLGR)`

Input: A record t_i and a line l_j
Output: Segmentation t_j^{i*} of l_j that minimizes $d(t_i, t_j^{i*})$

1 $M[0,0] \leftarrow 0$
2 **for** *each* $w \in 1 \dots |l_j|$ **do**
3 $\quad M[0,w] \leftarrow \infty$
4 **for** *each* $p \in 1 \dots m$ **do**
5 $\quad M[p,0] \leftarrow M[p-1,0] + d(null, t_i[k])$
6 **for** *each* $p = 1 \dots m$ **do**
7 $\quad$ **for** *each* $w = 1 \dots |l_j|$ **do**
8 $\quad\quad M[p,w] = \min \begin{cases} M[p-1,x] + d(l_j[x+1 \dots w], t_i[p]) \\ \qquad\qquad \forall 0 \le x < w \\ M[p-1,w] + d(\text{null}, t_i[p]) \end{cases}$
9 $t_j^{i*} \leftarrow$ construct segmentation by back tracing $M[m, |l_j|]$
10 **Return** t_i

For every node x in the search graph, $A^\star$ keeps a cost function $f(x)$ to determine the order in which $A^\star$ visits the nodes in the graph. $f(x)$ consists of two components: $g(x)$ and $h(x)$. $g(x)$ is the currently known least cost from start node to x. $h(x)$ is an estimated cost from x to the end node. $h(x)$ must be *admissible*, i.e., $h(x)$ must underestimate the actual cost from x to the end node. $h(x)$ being admissible guarantees the the path $A^\star$ finds is the least cost path when terminated. $h(x)$ is said to be *monotonic* if it satisfies $L(X) + h(x) \le L(Y) + h(y)$, where X is any path from start node to x, and Y is the path extending X with an additional node y. If $h(x)$ is monotonic, any node only needs to be processed once, making the search more efficient.

F. PROOF OF LEMMA 1

Let t_i^Z be the segmentation corresponding to Z, t_i^X be the partial segmentation corresponding to X, and t_i^Y be the partial segmentation corresponding to Y, we have $t_i^Z[k] = t_i^X[k], \forall 1 \le k \le p$ and $t_i^Z[k] = t_i^Y[k], \forall p < k \le m$. Let t_j^Z be the record aligning l_j with t_i^Z, t_j^X be the record aligning l_j with t_i^X, and t_j^Y be the record aligning l_j with t_i^Y. we have (20):

$$\begin{aligned} L(Z) &= \sum_{\substack{1\le j\le n \\ j\neq i}} \sum_{1\le k\le m} d(t_i^Z[k], t_j^Z[k]) \\ &= \sum_{\substack{1\le j\le n \\ j\neq i}} \sum_{1\le k\le p} d(t_i^Z[k], t_j^Z[k]) + \sum_{\substack{1\le j\le n \\ j\neq i}} \sum_{p<k\le m} d(t_i^Z[k], t_j^Z[k]) \\ L(X) &= \sum_{\substack{1\le j\le n \\ j\neq i}} \sum_{1\le k\le p} d(t_i^X[k], t_j^X[k]) \\ L(Y) &= \sum_{\substack{1\le j\le n \\ j\neq i}} \sum_{p<k\le m} d(t_i^Y[k], t_j^Y[k]) \qquad (20) \end{aligned}$$

$$L(X) \le \sum_{\substack{1\le j\le n \\ j\neq i}} \sum_{1\le k\le p} d(t_i^Z[k], t_j^Z[k]) \qquad (21)$$

$$L(Y) \le \sum_{\substack{1\le j\le n \\ j\neq i}} \sum_{p<k\le m} d(t_i^Z[k], t_j^Z[k]) \qquad (22)$$

(21) holds because $t_i^Z[k] = t_i^X[k], \forall 1 \le k \le p$ and t_j^X is the

record that achieves the minimal distance with t_i^X according to the definition of $L(X)$, which is certainly smaller or equal to any other record's (including t_j^Z) distance with first p columns of t_i. (22) holds for similar reasons.

Combining (20) (21) (22), we have $L(X) + L(Y) \leq L(Z)$.

G. HEURISTIC FUNCTION

Algorithm 4 describes the procedure for calculating h function for every node $[p, w]$ in G_i. Lines 1-8 initialize $freeD(c_1)$ for every candidate column c_1 in l_i. Lines 9-13 initialize and take care of corner cases of $h_{l_i}(p, w)$. Lines 14-16 use a dynamic programming procedure to initialize all entries of $h_{l_i}(p, w)$ similar to the one used in Algorithm 3.

Algorithm 4: `Calculate` h `Function`

Input: List L, number of columns m, anchor line l_i
Output: $h_i(p, w)$ for $1 \leq p \leq m$ and $1 \leq w \leq |l_i|$

1 **for** *candidate column* $c_1 \in l_i$ **do**
2 $freeD(c_1) \leftarrow 0$
3 **for** *Line* $l_j \in L$ *and* $l_j \neq l_i$ **do**
4 $freeD(c_1, l_j) \leftarrow$ Double.Max
5 **for** *candidate column* $c_2 \in l_j$ **do**
6 **if** $d(c_1, c_2) < freeD(c_1, l_j)$ **then**
7 $freeD(c_1, l_j) \leftarrow d(c_1, c_2)$
8 $freeD(c_1) \leftarrow freeD(c_1) + freeD(c_1, l_j)$
9 $h_{l_i}(m, |l_i|) \leftarrow 0$
10 **for** $w = |l_i| - 1 \rightarrow 1$ **do**
11 $h_{l_i}(m, w) \leftarrow \infty$
12 **for** $p = m - 1 \rightarrow 1$ **do**
13 $h_{l_i}(p, |l_i|) \leftarrow h_{l_i}(p + 1, |l_i|) + freeD(\text{null})$
14 **for** $p = m \rightarrow 1$ **do**
15 **for** $w = |l_i| \rightarrow 1$ **do**
16

$$h_{l_i}(p, w) = \min \begin{cases} h_{l_i}(p + 1, x) + freeD(l_i[w + 1 \dots x]) \\ \qquad\qquad \forall w < x \leq |l_i| \\ h_{l_i}(p + 1, w) + freeD(\text{null}) \end{cases}$$

17 **Return** $h_i(p, w)$

Proof of Lemma 2. h function is admissible by definition since for every candidate column s_1 in l_i, it finds the best candidate column in every other line to align with s_1.

According to Line 16 in Algorithm 4, we have:

$$h(p, w) \leq h(p + 1, w') + freeD(l_i[w + 1 \dots w']) \quad (23)$$

Let t_i^X be the segmentation corresponding to X and $t_i^{X'}$ be the segmentation corresponding to X'. We have $t_i^X[k] = t_i^{X'}[k], \forall 1 \leq k \leq p$. For every other line l_j, let t_j^X be the record of aligning l_j with t_i^X and $t_j^{X'}$ be the record of aligning l_j with $t_i^{X'}$. We have the following:

$$L(X) = \sum_{\substack{1 \leq j \leq n \\ j \neq i}} \sum_{1 \leq k \leq p} d(t_i^X[k], t_j^X[k])$$

$$L(X') = \sum_{\substack{1 \leq j \leq n \\ j \neq i}} \sum_{1 \leq k \leq p+1} d(t_i^{X'}[k], t_j^{X'}[k])$$

$$= \sum_{\substack{1 \leq j \leq n \\ j \neq i}} \sum_{1 \leq k \leq p} d(t_i^{X'}[k], t_j^{X'}[k]) + \sum_{\substack{1 \leq j \leq n \\ j \neq i}} d(t_i^{X'}[p + 1], t_j^{X'}[p + 1]) \quad (24)$$

Similar to the argument in Proof F, we have 25.

$$L(X) \leq \sum_{\substack{1 \leq j \leq n \\ j \neq i}} \sum_{1 \leq k \leq p} d(t_i^{X'}[k], t_j^{X'}[k]) \quad (25)$$

Because $l_i[w+1 \dots w'] = t_i^{X'}[p+1]$, we have (26) for every line l_j from the definition of $freeD$.

$$freeD(l_i[w + 1 \dots w'], l_j) \leq d(t_i^{X'}[p + 1], t_j^{X'}[p + 1]) \quad (26)$$

Summing up (26) for every $j \neq i$, we have:

$$freeD(l_i[w + 1 \dots w']) \leq \sum_{\substack{1 \leq j \leq n \\ j \neq i}} d(t_i^{X'}[p + 1], t_j^{X'}[p + 1]) \quad (27)$$

Combining Equations (23) (24) (25) (27), we have:

$$L(X) + h(p, w) \leq L(X') + h(p + 1, w')$$

H. JACCARD DISTANCE

Since Jaccard distance also reflects the strength of co-occurrence between s_1, s_2, we also used Jaccard for semantic distance. Recall that $JaccardDist(s_1, s_2) = 1 - \frac{|C(s_1) \cap C(s_2)|}{|C(s_1) \cup C(s_2)|}$.

While using Jaccard distance also achieves reasonable quality, its performance is not as good as using NPMI. We suspect that the reason lies in the fact that Jaccard cannot handle asymmetric sets well. In particular, suppose s_1 is very popular (e.g. "USA") but s_2 is not (e.g., "Congo"). When using Jaccard, just because s_1 is so popular, even if s_2 always co-occurs with s_1 the Jaccard similarity is low and Jaccard distance is high. In comparison, NPMI is information theoretic and handles asymmetry better – even if s_2 is not popular, as long as s_1 and s_2 always co-occur their NPMI will never be negative. Thus NPMI is more robust to asymmetric sets and is a better measure to use in our setting.

I. DETAILS OF SYNTACTIC DISTANCE

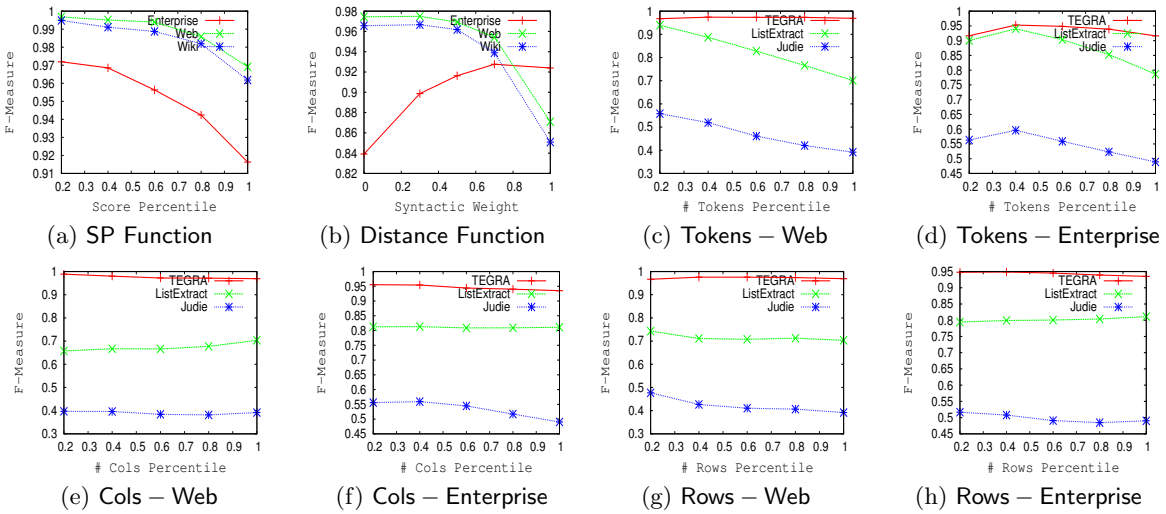
At a high level, our syntactic distance $d_{syn}(s_1, s_2)$ is

$$d_{syn}(s_1, s_2) = \frac{d_{len}(s_1, s_2) + d_{char}(s_1, s_2) + d_{type}(s_1, s_2)}{3}$$

We use three main components to capture syntactic distance. First, length difference is defined as $d_{len}(s_1, s_2) = \frac{||s_1| - |s_2||}{\max(|s_1|, |s_2|)}$, where $|s_1|$ denotes the number of tokens in s_1. One can prove that d_{len} satisfies triangular inequality. Second, character distance $d_{char}(s_1, s_2)$ looks at s_1 and s_2 in character level, and determines if they have same number of digits, capital letters, punctuation marks, symbols, and letters. $d_{char}(s_1, s_2)$ equals to the number of types of characters s_1 and s_2 have the same number, divided by the total types of characters, which is 5. Third, type distance indicates $d_{type}(s_1, s_2)$ whether s_1 and s_2 belong to the same predefined type, e.g., numerical value, phone number, email address, all of which determined by predefined regular expressions. Define $d_{type}(s_1, s_2)$ to be 0 when they s_1 and s_2 have the same type, and 1 otherwise. It can be shown that $d_{syn}(s_1, s_2)$ satisfies triangle inequality.

Note that when calculating the distance between a string s and a null string, i.e., $d(\text{null}, s)$, we treat null as an empty string for calculating the syntactic distance $d_{syn}(\text{null}, s)$, i.e., we use $d_{syn}(\text{“”}, s)$ for $d_{syn}(\text{null}, s)$; we give 1.0 for the semantic distance $d_{sem}(\text{null}, s)$ since empty string does not carry any meaningful semantics.

Also note that our design $d(s_1, s_2)$ did not include the likelihood of s_1 or s_2 being a cell in a table, called single value distance, which also seems to be a valid component of $d(s_1, s_2)$. Appendix J provides an explanation about this.



(a) SP Function (b) Distance Function (c) Tokens – Web (d) Tokens – Enterprise

(e) Cols – Web (f) Cols – Enterprise (g) Rows – Web (h) Rows – Enterprise

Figure H.1: Experiments in supervised setting

Lastly, our syntactic distance do not work well for lists with very long texts (e.g., paragraphs). In practice, very long lists are unlikely to have good relational tables, so we simply discard lists whose lengths exceed certain limit. Images and other HTML constructs are sometimes embedded in lists. These however can be easily identified by HTML tags and are removed from the input lists by an upstream table/list extraction job.

J. SINGLE VALUE DISTANCE

$d(s_1, s_2)$ reflects the likelihood of s_1, s_2 being in the same column of a table, which include $d_{syn}(s_1, s_2)$ and $d_{sem}(s_1, s_2)$. $d(s_1, s_2)$ could also include two additional components: the likelihood of s_1 being a cell in a table $d(s_1)$ and the likelihood of s_2 being a cell in a table $d(s_2)$. $d(s_1)$ could include the language model of s_1, the frequency of s_1 in a column of the web tables corpus, etc. We chose not to include $d(s_1)$ and $d(s_2)$ in our design of $d(s_1, s_2)$ for several reasons: (1) we found that the $d(s_1)$ and $d(s_2)$ is already partially expressed by $d(s_1, s_2)$. For instance, if s_1 occurs infrequently in the web tables corpus, so will the occurrence of s_1, s_2; (2) it is quite difficult to tune the relative weights of four components, $d_{syn}(s_1, s_2)$, $d_{sem}(s_1, s_2)$, $d(s_1)$, and $d(s_2)$; and (3) we found in experiments that bad relative weight assignment of the four components will greatly degrade the tables quality. Furthermore, the best quality of resulted tables after fine turning the relative weights is not so much better than our current much simpler distance function design.

K. MORE EXPERIMENTS

We also experiment with the supervised scenario where users need to segment an ad-hoc list into tables, and example row segmentations can be provided. By default we use two example segmentations as input.

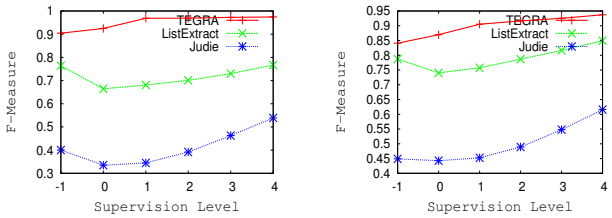
Table 7 varies the background corpus, where we have similar observations as Table 6 in the unsupervised setting.

Figure H.1 reports the same set of experiments as ones in Figure 8, but this time with supervision. We observe similar trends in the supervised setting.

Figure K.1 analyzes how quality improves as more user feedback is provided. The leftmost point ($x = -1$) corresponds to the unsupervised scenario where lists are segmented fully automatically. In the next set of points ($x = 0$), the correct number of columns are given as input. As we move to the right with $x > 0$ number of fully segmented rows as input, quality numbers further improve. We observe that

Test-Dataset	Background	TEGRA	ListExtract	Judie
Web	B-Web	**0.97**	**0.70**	0.39
	B-Enterprise	0.79	0.67	0.39
	B-Combined	**0.97**	**0.70**	0.39
Wiki	B-Web	**0.96**	**0.71**	0.50
	B-Enterprise	0.79	0.69	0.50
	B-Combined	**0.96**	**0.71**	0.50
Enterprise	B-Web	0.92	0.79	0.49
	B-Enterprise	0.92	**0.83**	0.49
	B-Combined	**0.94**	0.81	0.49

Table 7: F-Measure using different background corpus (supervised)



(a) Web (b) Enterprise

Figure K.1: Varying supervision level

TEGRA gets a significant boost with only 1 example row, and the incremental gain of have more example segmentations quickly diminishes. This shows that in the supervised setting TEGRA can achieve very high quality with very limited feedback. It is somewhat surprising that for `ListExtract`, knowing the number of columns ($x = 0$) actually hurts its quality. We suspect that the reason lies in the fact that `ListExtract` splits lines using a language model, which favors shorter token sequences (e.g., "New York" over "New York City"). As such, it tends to be too aggressive and over-segment. When the number of columns are constrained, however, the local splitting decisions cannot be reconciled by using more number of columns, thus destroying the structure of the extracted table and negatively affecting performance.