Weaving Text into Tables

Dhruv Gupta
Max Planck Institute for Informatics
Saarland Informatics Campus, Germany

Klaus Berberich
Max Planck Institute for Informatics, Germany
htw saar, Germany

ABSTRACT

In this paper, we showcase JIGSAW, a system that is able to shape unstructured text into structured tables for user-defined schemas. In short, to structure text into tables, JIGSAW leverages the lexico-syntactic structure imposed by linguistic annotations (e.g., part-of-speech, named entities, temporal and numerical expressions) on natural language text. We describe how challenging knowledge-centric tasks such as question answering, summarization, and analytics can be greatly simplified with the help of JIGSAW.

ACM Reference Format:
Dhruv Gupta and Klaus Berberich. 2020. Weaving Text into Tables. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20), October 19–23, 2020, Virtual Event, Ireland*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3340531.3417442

1 INTRODUCTION

Structured information present in the form of tables is invaluable for knowledge-centric tasks such as question answering, summarization, and analytics. However, most of the tabular information present on the Web or within documents is manually curated. To manually assemble tables is a labor-intensive process that involves: going through multiple documents relevant to the query; spotting relevant fact-bearing sentences; and finally tabulating the key facts. In this work, we demonstrate how JIGSAW automates this process of tabulating information from text in a query-driven manner, within seconds, and at scale across millions of documents.

JIGSAW structures text into tables with the help of the lexico-syntactic structure imposed by linguistic annotations. Nowadays, large document collections can be reliably annotated for part-of-speech tags, named entities, temporal, and numerical expressions. In a nutshell, JIGSAW structures text into tables in three steps. First, the user defines the schema of the table to be generated from the document collection by specifying a structured query pattern consisting of word sequences and annotations. Second, based on this structured query pattern we leverage GYANI [8], our indexing infrastructure for annotated documents, to extract text regions that match the structured query pattern and helps JIGSAW put together an initial raw table. Third and finally, JIGSAW operates on the initial raw table as input and performs linking of near-duplicate rows, resolution of NULL values using the document context, and computes a final table to be presented to the user.

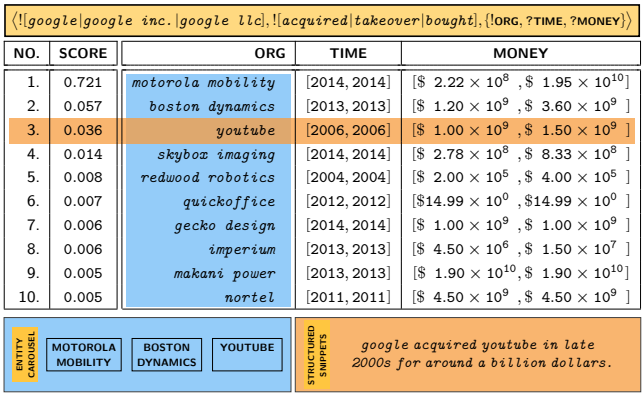

Figure 1: An example table generated by JIGSAW for the query concerning acquisitions made by Google from the Gdelt document collection. The structured query contains the bindings for which the resulting table is populated using the indexed document collection. The table can then be used to generate popular information objects such as entity carousels and structured snippets.

Our system JIGSAW [9], is able to generate structured tables from unstructured documents given a schema as an input. JIGSAW offers the capability to present partial, paraphrased, and redundant knowledge spread across millions of documents into structured tables. This ability allows us to perform knowledge-centric analytics over large document collections; a capability that was only possible over relational data present in traditional database management systems. In this work, we demonstrate JIGSAW's capability to structure text into tables and describe three use cases that can benefit from it, namely: question answering, summarization, and analytics.

Organization. The remainder of the article is structured as follows. In Section 2, we describe the internal architecture of JIGSAW and the query language that is offered to the user to describe the table's schema. In Section 3, we describe the data, software, and hardware setup of our system JIGSAW. In Section 4, we describe the applications of question answering, summarization and analytics that will benefit from JIGSAW's capabilities. Finally, we conclude the article with a survey of prior art and a summary of the capabilities that JIGSAW offers to the research community.

2 JIGSAW ARCHITECTURE

To generate structured tables from text documents JIGSAW leverages the lexico-syntactic structure that annotations help impose on natural language text. We first describe, in brief, the data model and indexes that allow us to perform structured search over the annotated document collection. Second, we describe the query language that allows the user to define the schema of the table to be generated from text documents. Third and finally, we explain how the initial raw table is shaped into the final table by linking redundant rows, resolving NULL values, and computing representative rows.

CIKM '20, October 19–23, 2020, Virtual Event, Ireland

ACM ISBN 978-1-4503-6859-9/20/10.
https://doi.org/10.1145/3340531.3417442

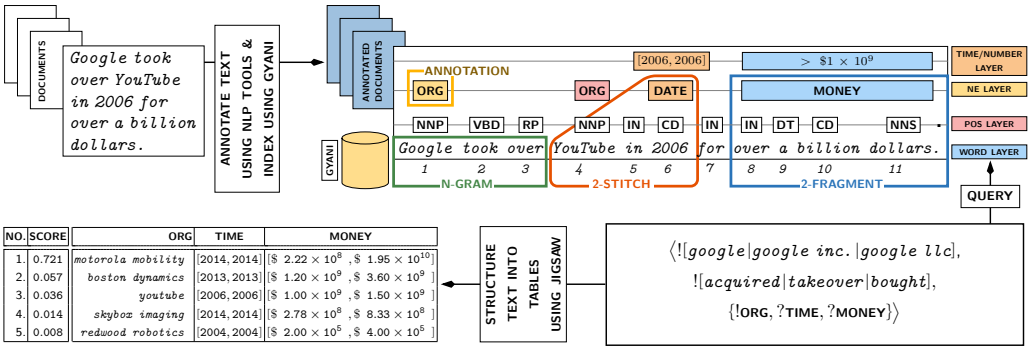

Figure 2: Overview of JIGSAW. The figure shows how the text documents are annotated and the indexing units that are stored in our indexing infrastructure. Furthermore, an example query shows how to define the table schema that simultaneously defines that structure of the sentences to retrieve from the annotated document collection. The table generated from the document collection is shown at the end.

2.1 Annotated Document Model and Indexes

To perform structured search over text we take the assistance of semantic annotations, such as part-of-speech, named entities, temporal, and numerical expressions, that modern natural language processing (NLP) tools can deliver with reliable accuracy. To represent the semantic annotations that mark up the word sequences, we arrange them in a series of layers (see Figure 2). We begin with the word layer that contains the sequence of words along with their positions in the document. Building on the word layer, we place annotations that share the same semantics in different layers with positional information that indicate what word sequences they adorn. For instance, in Figure 2, MONEY annotates the word sequence *over a billion dollars* spanning the positions [8, 11] and is placed in the named entity (NE) layer. Based on this data model, we create a set of five indexes that store combinations of word sequences and annotations. First, we create N-GRAM and SKIP-GRAM INDEXES that store unigrams, bigrams, trigrams, and skip-grams (within a window of ten words) derived from the word layer. Additionally, for these indexing units we create N-GRAM and SKIP-GRAM DICTIONARIES that store their collection and document frequencies. Second, we create ANNOTATION INDEXES for each annotation type in the different layers. Next, we create indexes that stores combinations of n-grams and annotations. To this end, we create our third and fourth index types: 2-STITCH INDEXES that stores ordered combinations of n-grams and annotations and 2-FRAGMENT INDEXES that stores aligned combinations of n-grams and annotations. Fifth and finally, we create a DIRECT INDEX that stores the entire representation of the layered-document model. Example of each of the indexing units in the five types of indexes is illustrated in Figure 2. With the help of these indexes we can locate sentences that correspond to the structured query pattern that defines the table schema.

2.2 QUERY Operators

We now describe the query operators that help define the schema of the table to be generated from the document collection. To simplify the description of the table schema, we present a declarative query language over annotated text for table generation. A typical query for table description consists of word sequences, annotations, or a combination of both. In short, the query specifies the structure of the text region (which can be span a few words, sentences, or paragraphs) to search in the document collection(s). We next describe the key operators that assist in specifying this structure of the text region as well as the table. Figure 2 shows an example query that generates a table for acquisitions made by Google.

Binding Operators specify the word sequences and annotations that must be matched in our annotated data model to form the attributes of the table. The $\boxed{!\ell}$ operator is a *compulsory* bindings operator that specifies that the annotation ℓ must occur in the structure of the text region. Whereas, the $\boxed{?\ell}$ operator is a *relaxed* bindings operator that relaxes the match of the annotation ℓ when its occurrence is not found in the text region. When, the value can not be spotted the $\boxed{?\ell}$ bindings operator results in a NULL value to be filled in the cell value for that attribute. **Markers** such as the *union*, *wildcard*, and *multiplicity* help specify additional semantics to the binding operators. The union marker (|) helps specify disjunctive semantics to multiple word sequences or annotations specified with the same binding operator. For instance, the binding ?[*united states* | *us*] looks for either the word sequence *united states* or *us* in the sentence; if none is found the cell value is filled with NULL. The wildcard marker $\boxed{*}$ specifies variable-length gaps to be expressed in the bindings for word sequences. For example, ?⟨*invested in* ∗ *company*⟩ matches and places the word sequences between *invested in* and *company* as cell values. The multiplicity marker $\boxed{\times\{m,n\}}$ specifies the minimum and maximum times an annotation can occur in a sequence. For instance, the binding sequence ⟨*google acquired*, ORG×{1,3}⟩ retrieves sentences that contain the word sequence *google acquired* and one or more but less than or equal to 3 mentions of an organization.

Stack Operator $\oplus$ allows the user to attach semantics to word sequences when querying for text regions. For instance, with the binding ?*2006* we will retrieve text regions that contain 2006 both as a numerical and as a temporal expression. However, with the binding ?(*2006*⊕TIME) we restrict the match to those text regions that contain the mention of 2006 as a temporal expression.

Order and Unorder Operators help specify the order in which its arguments are matched to retrieve text regions for table generation. The order operator $\boxed{\langle\bullet\rangle}$ helps specify schemas for *asymmetric relationships*. For instance, ⟨?*google*, ?[*acquired* | *bought*], ?ORG⟩ retrieves text regions that match all the bindings in the specified order. The unorder operator $\boxed{\{\bullet\}}$ helps specify schemas for *symmetric relationships*. For instance, {?PERSON, *married*, ?PERSON} disregards the order among the bindings during their retrieval.

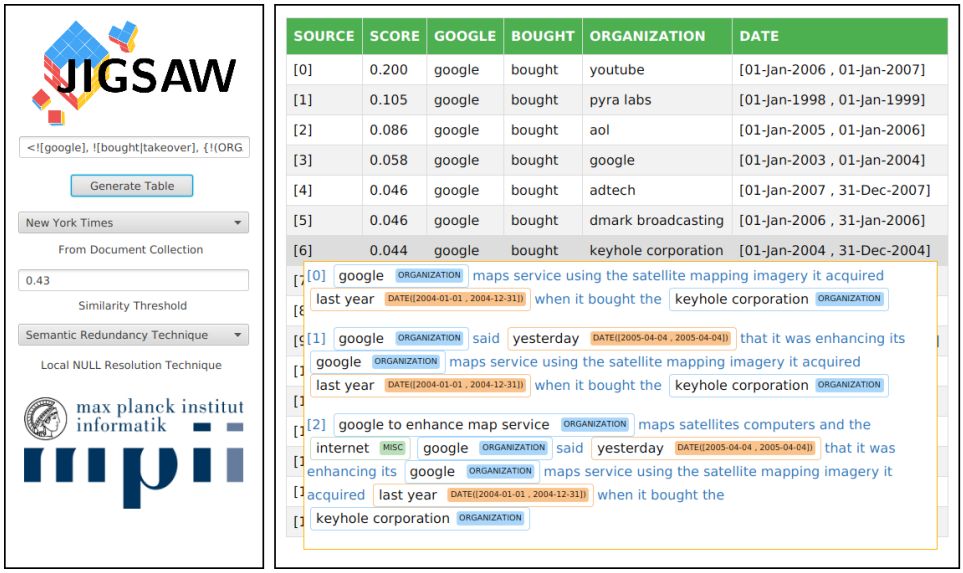

Figure 3: GUI for JIGSAW. The user can specify the table schema to be generated from the text documents in the search bar. The figure displays the table generated from the query $\langle ![google], ![bought \mid takeover], \{!\text{ORG}, ?\text{DATE}\}\rangle$ from the New York Times news archive. The resulting table additionally contains the sources which help derive the rows of the table. Hovering above the source cells reveals the text snippets as tooltips.

Assembling the Table. We leverage our indexing infrastructure GYANI to retrieve text regions that correspond to the structured query pattern. GYANI helps us create an initial raw table containing attributes (columns) that correspond to the bindings. While each row in the raw table contains the cell values corresponding to each retrieved text region. Next, we describe the various operators that help shape the final table to be presented to the user.

2.3 LINK Operators

We consider cell values in the tables to be of two types: text and numbers. Text cell values will arise when bindings for annotations are obtained from the word, part-of-speech, and named entity layer. While, the numerical cell values will arise when bindings for annotations are obtained from the time and number layers. Note that all resolved values for numerical and temporal expressions are modeled as intervals in their respective layers.

LINK Operators define how the redundant rows in the raw table will be linked together. This is done by computing an average of surface, contextual, and semantics based similarities among the cell values between two rows. First, we compute the surface level similarity as an edit-distance based similarity between text values and equality between numerical values. Second, we compute contextual similarity by using the Jaccard coefficient over the bag-of-words derived from the source text region for text values and interval overlaps for numerical values. Third, we compute semantic similarity based on how frequently words in the text-based cell values co-occur in the entire corpus. Semantic similarity for numerical values is computed using Jaccard-overlap of intervals that model uncertainty indicated by textual modifiers (e.g., in the numerical expression *around a billion dollars*, "around" is a textual modifier). These similarities compute a score indicating a degree of similarity between the raw rows. A threshold on the similarity value can be set by the user that considers two raw rows as being near duplicates.

Local NULL Value Resolution. The $?\ell$ binding operator may result in NULL cell values. We use a set of three techniques to resolve these NULL values, namely: scoping, proximity, and semantic redundancy. In all the NULL value resolution techniques, we make an estimate from the document in which the text region resides. In the scoping technique, we take the most frequent annotation as an estimate. In the proximity technique, we take the annotation that lies closest to the matched text region within the document. In the semantic redundancy technique, we take the most frequent annotation in their respective semantic models for making an estimate. Depending on query intent, certain NULL resolution techniques work better than others. If the query is entity oriented then the proximity based technique performs well as it functions like a co-reference resolution method. However, if the query is event related then the semantic redundancy performs better.

2.4 ANALYSIS Operators

We next discuss the operators that help to shape the final table. The ANALYSIS operators allow us to flatten a group of linked rows, score the representative rows, and rank them.

FLAT Operator takes a set of linked raw rows and computes a representative row to be added to the final table. This can be done in two ways: piecing together a representative row or selecting the most similar row from the set. By piecing together a representative row, we select the attribute value from the set of raw rows that is the most similar to remaining values in the set. By doing this for all the table attributes, we can construct the representative row to be part of the final table. Alternatively, we can select a row that is most similar to the others in its entirety to be part of the final table. However, with the piecewise construction of the representative row, we can resolve those NULL values, which could not be estimated using the local NULL resolution technique, with the help of the other linked rows. We refer to this as **global NULL resolution**.

Table 1: Statistics for the semantically annotated document collections.

COLLECTION	SIZE (GB)	#DOCUMENTS	#WORDS	#SENTENCES	#PART-OF-SPEECH	#NAMED ENTITY	#TIME	#NUMBERS
NYT	49.7	1,855,623	1,058,949,098	54,024,146	1,058,949,098	107,745,696	15,411,681	21,720,437
WIKIPEDIA	156.0	5,327,767	2,807,776,276	192,925,710	2,807,776,276	444,301,507	97,064,344	82,591,612
GIGAWORD	193.6	9,870,655	3,988,683,648	181,386,746	3,988,683,648	517,420,195	72,247,124	102,299,554
GDELT	296.2	14,320,457	6,371,451,092	297,861,511	6,371,451,092	640,812,778	94,009,542	104,964,085

Table 2: Sizes of indexes and dictionaries in Gigabytes (GB).

INDEX TYPE	NYT	WIKIPEDIA	GIGAWORD	GDELT
N-GRAM DICTIONARIES	4.54	19.80	10.50	19.04
SKIP-GRAM DICTIONARY	14.40	25.70	21.30	29.30
N-GRAM INDEXES	45.90	126.30	154.40	234.80
SKIP-GRAM INDEX	56.10	180.80	203.60	289.00
ANNOTATION INDEXES	2.39	7.65	9.33	16.03
2-FRAGMENT INDEXES	6.30	23.10	24.16	36.84
2-STITCH INDEXES	141.00	473.00	542.40	677.10
DIRECT INDEX	18.80	44.80	52.40	82.30

`SCORE` **Operator** assigns a score to the representative rows using: support (equal to the ratio of number of linked raw rows to total number of raw rows) of the row and its novelty among the other rows in the final table.

`RANK` **Operator** assigns a ranking to the rows of the final table. We provide two ways of ranking: by the generated scores and by average inverse length of text regions supporting a representative row.

3 JIGSAW IMPLEMENTATION

Our indexing infrastructure GYANI and JIGSAW were both implemented in Java from scratch. The graphical user interface (GUI) for the system is also implemented in Java using the JavaFX GUI framework. We instantiate our indexing infrastructure in HBase, a distributed and extensible record store that runs on our Hadoop cluster of twenty server-grade machines. We create our indexes and dictionaries for four different document collections: the New York Times Annotated corpora [2], the Fifth Edition of English Gigaword [3], Gdelt news archive [1], and the English Wikipedia [4]. The details of the four semantically annotated document collections and their index sizes are given in Tables 1 and 2 respectively.

4 APPLICATIONS

We now discuss three use cases that will benefit from JIGSAW's table generation capabilities.

Question Answering. Users interacting with search systems are no longer interested in going through long lists of documents to find answers to their queries. Most users' queries nowadays are directly answered by structured snippets [5], knowledge panels [10], and entity carousels [13]. However, most state-of-the-art information retrieval (IR) systems can only present such information elements derived from resources that are *already* structured, for example, in the form of lists, tables, and Wikipedia infoboxes. Data in the form of tables models all the possible *information objects* [7] that can be derived from text documents. Structured tables obtained from JIGSAW present a summary of important information drawn from multiple documents. For example, the table shown in Figure 1 contains acquisitions made by Google extracted from the Gdelt news archive. From this table, we can generate *entity carousels* for the query `google acquisitions` by taking the ORG column and displaying entities relevant for the query. Furthermore, we can generate *structured snippets* by taking a row (e.g., ⟨ YOUTUBE, [2006, 2006], [$ 1.00 × 10^9, $ 1.50 × 10^9]) ⟩) and displaying the shortest evidence that helped generate the row.

Summarization. Structured tables present a summary of important information drawn from multiple documents. Moreover, with JIGSAW's capability to generate tables, we can link facts from different document collections thus offering to contextualize the information from multiple points of view. For instance, by generating tables for the same schema from Wikipedia and the New York Times news archive we can help link facts mentioned in the encyclopedia with fine granular information of events present in the news articles.

Analytics. In order to derive any meaningful insight from massive amounts of textual data, a preprocessing step is required to format the data so that it can be processed by machine learning methods. Often, this representation is in the form of a table. With JIGSAW's capabilities we can quickly assimilate facts and figures in the form of tables that allow for complex machine learning models to discover insights (e.g., outlier detection).

5 RELATED WORK

We now discuss related systems that precede JIGSAW in providing semantic and structured search capabilities over annotated document collections. [6] provides a combined search capability over knowledge graphs and document collections with a query language that allows searching for keyword queries along with semantic classes and entities. [11, 12] take a different approach to semantic search, by linking entities in text to knowledge graphs. Thereafter, the authors provide analytic visualizations to be created on frequency of entities with document's publication date. [14] propose a LOAD graph representation of annotated document collection for enabling semantic search. In contrast to these approaches, our data model can accommodate any kind of semantic annotation as well as provide the flexibility to create structured tables for any user-defined schema.

REFERENCES

[1] The GDELT Project. `https://www.gdeltproject.org/`.
[2] The NYT Corpus. `https://catalog.ldc.upenn.edu/LDC2008T19`.
[3] English Gigaword. `https://catalog.ldc.upenn.edu/LDC2011T07`.
[4] Wikipedia: The Free Encyclopedia. `https://www.wikipedia.org/`.
[5] Structured Snippets in Google Web Search. `https://ai.googleblog.com/2014/09/introducing-structured-snippets-now.html`.
[6] Bast H. et al. 2014. Semantic Full-Text Search with Broccoli. In SIGIR'14.
[7] Culpepper J. S. et al. Research Frontiers in Information Retrieval: Report from the Third Strategic Workshop on Information Retrieval in Lorne (SWIRL 2018). *SIGIR Forum* 52, 1 (2018), 34–90.
[8] Gupta D. and Berberich K. GYANI: An Indexing Infrastructure for Knowledge-Centric Tasks. In CIKM'18.
[9] Gupta D. and Berberich K. JIGSAW: Structuring Text into Tables. In ICTIR'19.
[10] Henry J.W. 2013. Providing Knowledge Panels with Search Results. US Patent App. 13/566,489.
[11] Hoffart J. et al. AESTHETICS: Analytics with Strings, Things, and Cats. In CIKM'14.
[12] Hoffart J. et al. STICS: Searching with Strings, Things, and Cats. In SIGIR'14.
[13] Hong P. J. et al. 2018. Related Entities. US Patent App. 15/798,175.
[14] Spitz A. et al. EVELIN: Exploration of Event and Entity Links in Implicit Networks. In WWW'17.