JIGSAW: Structuring 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

We present JIGSAW, an end-to-end query driven system that efficiently generates structured tables from unstructured documents. To do so, first we describe how we can quickly retrieve sentences in support of structured queries that describe the table schema. Second, we describe how we can estimate table cell values using document context where such values can not be retrieved. Third, we describe how we can link together similar rows, rank, and diversify them to generate high-quality tables. We show that JIGSAW can generate tables from 25 million documents within seconds.

ACM Reference Format:
Dhruv Gupta and Klaus Berberich. 2019. JIGSAW: Structuring Text into Tables. In *The 2019 ACM SIGIR International Conference on the Theory of Information Retrieval (ICTIR '19), October 2–5, 2019, Santa Clara, CA, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3341981.3344228

1 INTRODUCTION

Tables are structured summaries obtained from multiple documents. Tables already present in documents or web tables are an important resource for tasks such as question answering, fact checking, and analytics. Manually generating tables is a laborious task. Teams of journalists often collaborate to curate tables using spreadsheet tools (e.g., Google Fusion Tables) [21]. To reduce this human effort, we need an information retrieval (IR) system that instead of presenting ten blue links, generates structured tables in response to queries.

Results in the form of structured snippets [5], knowledge panels [23], and lists of related entities [24] are gaining prominence in search results. To create them, structured data in the form of knowledge graphs (KGs) [4, 17] and web tables [13, 32] are leveraged. These approaches are limited as they can only use fixed schema associated with individual web tables or KG schema in the form of $\langle$S,P,O$\rangle$ triples. To generate tables for user-defined schema from unstructured text collections, we leverage semantic annotations that natural language processing (NLP) tools can now provide accurately. Concretely, annotations in the form of part-of-speech (e.g., $\texttt{google}\oplus$NNP), named entities (e.g., $\texttt{larry page}\oplus$PERSON), temporal (e.g., *2020s*$\oplus$[2020, 2029]), and numerical expressions (e.g., $\texttt{a million dollars}\oplus\$1 \times 10^6$) help us impose a lexico-syntactic structure over unstructured text. Using this insight, we can perform structured search over large document collections and put together tables using redundant, partial, and paraphrased pieces of text spread across millions of documents. Fig. 1 shows an example table containing Google acquisitions generated by JIGSAW.

ICTIR '19, October 2–5, 2019, Santa Clara, CA, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6881-0/19/10...$15.00
https://doi.org/10.1145/3341981.3344228

$\langle ![\textit{google}|\textit{google inc.}|\textit{google llc}], ![\textit{acquired}|\textit{takeover}|\textit{bought}], \{!\text{ORG}, ?\text{TIME}, ?\text{MONEY}\}\rangle$

NO.	SCORE	ORG	TIME	MONEY
1.	0.721	*motorola mobility*	[2014, 2014]	[$\$\ 2.22 \times 10^8$, $\$\ 1.95 \times 10^{10}$]
2.	0.057	*boston dynamics*	[2013, 2013]	[$\$\ 1.20 \times 10^9$, $\$\ 3.60 \times 10^9$]
3.	0.036	*youtube*	[2006, 2006]	[$\$\ 1.00 \times 10^9$, $\$\ 1.50 \times 10^9$]
4.	0.014	*skybox imaging*	[2014, 2014]	[$\$\ 2.78 \times 10^8$, $\$\ 8.33 \times 10^8$]
5.	0.008	*redwood robotics*	[2004, 2004]	[$\$\ 2.00 \times 10^5$, $\$\ 4.00 \times 10^5$]

Figure 1: An example table generated from the GDELT news archive.

2 PROBLEM DEFINITION

JIGSAW generates a table, given its schema, from large annotated document collections. As input, we are given a **structured query** $\mathcal{Q}$ that describes the table schema:

$$\text{Structured Query:} \quad \mathcal{Q} = \langle a_1, a_2, \ldots, a_N \rangle, \tag{1}$$

where, each **attribute** a can be specified with the help of word sequences (e.g., $\langle\texttt{took over}\rangle$), annotations (e.g., MONEY) or a combination of both word sequences and annotations (e.g., $\langle\texttt{youtube}\rangle\oplus$ORG). Let Σ_i denote the domain of values that attribute $a_i \in \mathcal{Q}$ can take. The **table schema** $\mathcal{R}$ is then defined by the query attributes. Concretely, a row $r \in \mathcal{T}$ has the following structure:

$$\text{Row Structure:} \quad r \subset 2^{\Sigma_1} \times 2^{\Sigma_2} \times \ldots \times 2^{\Sigma_N}. \tag{2}$$

In Eq. 2, an attribute of query $a_i \in \mathcal{Q}$ can refer to multiple elements from its domain Σ. Thus, tables generated by our approach can contain cells with multiple values (zero normal form). To construct the **final table** $\mathcal{T}$ we proceed in two steps. First, we construct a **raw table** $\overline{\mathcal{T}}$. To construct $\overline{\mathcal{T}}$, we retrieve text regions that match the query template $\mathcal{Q}$. Using the retrieved text regions we create a raw table that is a collection of rows that contain document metadata $d_{\texttt{id}}$, text region span in document $\texttt{text}(\cdot)$, and value(s) $\overline{c}$ (for simplicity consider singular values in Eq. 3) for the query template:

$$\text{Raw Table:} \quad \overline{\mathcal{T}} = \bigcup \overline{r} = \bigcup \big(\, d_{\texttt{id}}, \texttt{text}(\cdot), \overline{c}_1, \overline{c}_2, \ldots, \overline{c}_N \, \big). \tag{3}$$

In the raw table $\overline{\mathcal{T}}$, we allow for relaxed matches to the query template that will result in **unknown values** (**NULL**) as cell values. We resolve NULL values using two approaches: local and global resolution. Local resolution uses the document which establishes the provenance of the row to determine the missing values. Whereas, global resolution relies on other similar rows in the raw table (thereby leveraging cross-document evidences) to infer NULL values.

Second, having generated the raw table $\overline{\mathcal{T}}$, we link and aggregate rows that mention similar text, entities, relations, temporal or numerical values by leveraging the semantics of annotations in the cell values to arrive at the final table $\mathcal{T}$. Each aggregated row in the final table is assigned a $\texttt{score}(r)$ that reflects its prominence in the document collection. Furthermore, the rows in table $\mathcal{T}$ can be **ranked** using its originating context (provenance) $\texttt{rank}(r)$ and diversified amongst other rows in the table $\texttt{diversify}(r)$. Structure of the final table can now be defined as:

$$\mathcal{T} = \bigcup r = \bigcup \big(\, \cup\, d_{\texttt{id}}, \cup\, \texttt{text}(\cdot), \texttt{score}(\cdot), c_1, c_2, \ldots, c_N \, \big).$$

3 INDEXES OVER ANNOTATED TEXT

Annotated Text Model. Consider, a large document collection $\mathcal{D} = \{d_1, \ldots, d_{|\mathcal{D}|}\}$. Each document $d \in \mathcal{D}$ consists of sentences $d = \langle s_1, s_2, \ldots, s_{|d|} \rangle$ which further consist of a sequence of words $s = \langle w_1, w_2, \ldots, w_{|s|} \rangle$ drawn from the vocabulary of the collection $\Sigma_{\mathcal{V}}$. NLP tools can now deliver high-quality annotations over text in the form of parts-of-speech, named entities, numerical quantities, and temporal expressions. Consider, a NLP annotator $\mathcal{L}$ that further tags the sequence of words $\langle w_i \ldots w_j \rangle$ $(i \leqslant j)$ in documents with elements ℓ from its annotation alphabet $\Sigma_{\mathcal{L}}$. This way we obtain layers-of-annotation over text in documents (see Fig. 2). Semantic annotations help us impose a lexico-syntactic structure over text. This in turn helps us perform structured search over annotated document collections. To do so, we leverage GYANI [22] as our backend indexing infrastructure. We briefly describe the indexes we use to support our query operators.

Text Indexes help locate and compute statistics for text. To this end, we first create N-GRAM INDEXES and DICTIONARIES. N-GRAM indexes record unigrams, bigrams, and trigrams with their positional spans. While, the dictionaries record the document frequency (df) and collection frequency (cf) of n-grams. The n-grams are derived from the sentences in each document of $\mathcal{D}$. Furthermore, to speed up the retrieval of surface forms that are slight variations of a complete label (e.g., [`youtube video website` | `youtube website`]) we create SKIP-GRAM INDEXES and DICTIONARIES. Skip-gram indexes record ordered co-occurrences of words within a context of ten words. **Annotation Indexes** record for each annotation layer its element and positional span (e.g., MONEY with span $[8, 11]$ in Fig. 2). **Annotated Text Indexes** record positional spans for pair-wise and ordered combinations of word sequences and annotations. For each such combination in a sentence of a document, we create two indexes: 2-FRAGMENT and 2-STITCH indexes. Where, 2-FRAGMENTS are annotated word sequences (e.g., $\langle$`Google`$\oplus$ ORG$\rangle$ in Fig. 2) and 2-STITCHES are ordered co-occurrences of word sequences with other annotations in the sentences (e.g., $\langle$`YouTube`, DATE$\rangle$ in Fig. 2).

Direct Index. The inverted indexes help us retrieve positional spans corresponding to text regions for a given structured query. However, to retrieve the resolved annotation values (e.g., the resolved annotation value $> \$1 \times 10^9$ for the phrase `over a billion dollars`) we need to access the different layers of annotation for a given text region. To this end, we store the documents with all annotation layers and sentence boundaries in a DIRECT index.

4 QUERY OPERATORS

We next describe the operators that help define the table schema.

Binding Operator ($\boxed{!\ell}$ & $\boxed{?\ell}$) are unary operators that specify the annotations or word sequences that should be part of the table schema. The binding operator can be specified using annotations, e.g., $\langle$!PERSON, !ORG $\rangle$ or word sequences, e.g., !$\langle$`acquired`$\rangle$. The $\boxed{!\ell}$ operator requires that the values *must* be present in the text regions being retrieved to populate a column in the table. Whereas, $\boxed{?\ell}$ operator can perform a relaxed query match. That is, the $\boxed{!\ell}$ operator can not result in NULL values, whereas the $\boxed{?\ell}$ can be relaxed to a NULL value. Semantics of $\boxed{!\ell}$ and $\boxed{?\ell}$ can be specified as:

$$\text{match}(\ell_{\langle i,j \rangle}, s) = \{s \in d \mid d \in \mathcal{D} \wedge \ell_{\langle i,j \rangle} \sqsubset s\},$$

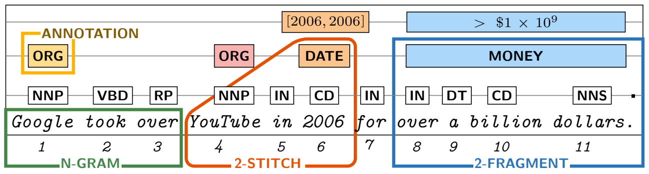

[2006, 2006]
> $1 × 10^9
ANNOTATION
ORG
ORG
DATE
MONEY
NNP VBD RP NNP IN CD IN IN DT CD NNS .
Google took over YouTube in 2006 for over a billion dollars.
1 2 3 4 5 6 7 8 9 10 11
N-GRAM
2-STITCH
2-FRAGMENT

Figure 2: Inverted indexes over the annotated text model.

where, $\ell_{\langle i,j \rangle}$ represents the text region $\langle w_i, \ldots, w_j \rangle$ tagged with annotation ℓ and $\sqsubset$ denotes that $\ell_{\langle i,j \rangle}$ is a contiguous subsequence in sentence s. Bindings can be decorated with markers to increase recall. These markers are: UNION, WILDCARD, and MULTIPLICITY.

UNION MARKER ($|$) helps to specify paraphrases for a word sequence binding, e.g., ![`acquired` | `takeover` | `buy out`]. The UNION marker applies the Boolean disjunctive semantics to the text regions matched for each of the paraphrases in the set.

WILDCARD MARKER ($\boxed{*}$) helps to indicate variable-length gaps in word sequences. With the $\boxed{*}$ marker, the corresponding cell values for the binding contain the text regions that fill in the wildcard. An example query using $\boxed{*}$ marker is: ?$\langle$`obtained` $\boxed{*}$ `award`$\rangle$.

MULTIPLICITY MARKER ($\boxed{\times\{m, n\}}$) helps to associate multiple annotation values in a cell value for the binding. The MULTIPLICITY marker specifies the minimum number m and maximum number n of annotation type ℓ a cell can contain for the binding. As an example query consider: $\langle$!$\langle$`google` $\boxed{*}$ `acquired`$\rangle$, !ORG $\boxed{\times\{1,3\}}\rangle$.

Stack Operator ($\oplus$) is a binary operator that helps in attaching additional semantics to word sequences, e.g., !$\langle$`paris`$\oplus$LOCATION$\rangle$. The semantics for the STACK $\oplus$ operator are:

$$\text{match}(w_{\langle i,j \rangle} \oplus \ell, s) = \{s \in d \mid d \in \mathcal{D} \wedge \ \ell_{\langle i,j \rangle} \sqsubset s \wedge w_{\langle i,j \rangle} \sqsubset s\},$$

where, the annotation ℓ and the word sequence $w_{\langle i,j \rangle}$ occupy the same positional span $[i, j]$ in the sentence of a document $s \in d$.

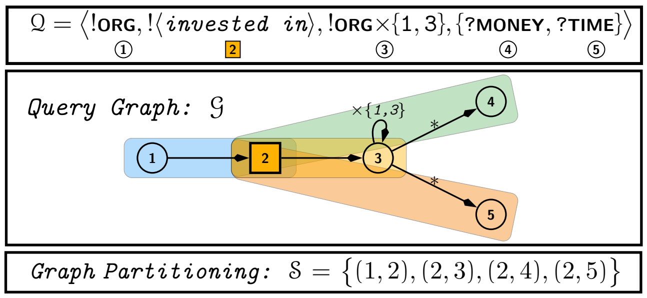
A structured query containing $\boxed{!\ell}$ or $\boxed{?\ell}$ will match annotated text regions that contain their arguments in an *ordered* sequence. A sequential order amongst the arguments of the query $a \in \mathcal{Q}$ helps to curate tables for **asymmetric relations**. For instance, in the table for the query, $\langle$!ORG$_1$, !$\langle$`has acquired`$\rangle$, !ORG$_2\rangle$, the matches for ORG$_1$ and ORG$_2$ can not be exchanged.

Unorder Operator ($\boxed{\{\bullet\}}$) is a n-ary operator that allows matching text regions irrespective of the order in which its arguments are mentioned. For example, the query, $\langle$!ORG, !$\langle$`has acquired`$\rangle$, {!ORG, ?MONEY, ?DATE}$\rangle$, treats the bindings for ORG, MONEY, and DATE in an unordered manner. The UNORDER operator is useful for creating tables for **symmetric relations** where the order amongst the bindings is not important, e.g., {!PERSON, !`married`, !PERSON}.

5 QUERY PROCESSING

We next discuss how to derive a query execution plan for retrieval of annotated text regions for a structured query $\mathcal{Q}$.

Query Graph. The structured query $\mathcal{Q}$, represents a template to be matched against the annotated text model. This sequence of query operators and their arguments can be succinctly represented in a graph. Let, $\mathcal{G}(V, E)$ represent a directed graph corresponding to the query $\mathcal{Q}$, where the set of vertices V represents the arguments (e.g., word sequences or annotation types) and E be a set of directed edges that represents the query operator semantics. We associate a function $q(e)$ with each edge $e \in E$, that defines either sequential, stacking, unorder, or multiplicity semantics between the connecting vertices. Fig. 3 shows an example of a query graph.


$\mathcal{Q} = \langle$!ORG, !$\langle$*invested in*$\rangle$, !ORG$\times\{1,3\}$, $\{$?MONEY, ?TIME$\}\rangle$
①
2
③
④
⑤
Query Graph: $\mathcal{G}$
1
2
3
4
5
$\times\{1,3\}$
*
*
Graph Partitioning: $\mathcal{S} = \{(1,2),(2,3),(2,4),(2,5)\}$
Figure 3: An example query, its graph, its partitioning, and assembly. Square nodes correspond to anchor vertices. Each graph partition $S \in \mathcal{S}$ is highlighted in color in the query graph $\mathcal{G}$.

Sequential Semantics inherent in the query structure $\mathcal{Q} = \langle a_1, a_2, \ldots, a_k \rangle$ are represented by a directed edge between two of the query arguments. Sequential semantics can also be specified with the help of the BINDINGS operator. The sequential semantics $q(e) \equiv q(a_i \to a_j)$ ensure that the mention of the argument a_i is before that of a_j in the annotated text model:

$$q(e) \equiv q(a_i \to a_j) = \Big\{ s \in d | d \in \mathcal{D} \wedge \ell^i_{\langle m,n \rangle} \in s \wedge \ell^j_{\langle p,q \rangle} \in s \wedge (m \leqslant n) \wedge (n < p) \wedge (p \leqslant q) \Big\},$$

where, $\ell^i_{\langle m,n \rangle}$ represents the annotation matching a_i and $\ell^j_{\langle p,q \rangle}$ represents the annotation matching a_j. The MULTIPLICITY marker constraints additionally imply that the number of argument value lie within bounds conveyed along with the marker.

Stacking Semantics specified by the STACK operator $q(e) \equiv q(a_i \xrightarrow{\oplus} a_j)$ conveys that the arguments to the $\oplus$ operator occupy the same positions in the sentence but adorn different annotation layers in the text model:

$$q(e) \equiv q(a_i \xrightarrow{\oplus} a_j) = \Big\{ s \in d | d \in \mathcal{D} \wedge \ell^i_{\langle m,n \rangle} \in s \wedge \ell^j_{\langle p,q \rangle} \in s \wedge (m \leqslant n) \wedge (p \leqslant q) \wedge (m = p) \wedge (n = q) \Big\}.$$

Unorder Semantics specified by the $\boxed{\{a_j, \ldots, a_k\}}$ operator $q(e) \equiv q(a_i \xrightarrow{*} \{a_j, \ldots, a_k\})$ specifies that any of the arguments specified by $a_j, \ldots, a_k$ can follow a_i in the annotated text model (for simplicity, we show only a_j from $\{a_j, \ldots, a_k\}$ below):

$$q(e) \equiv q(a_i \xrightarrow{*} a_j) = \Big\{ s \in d | d \in \mathcal{D} \wedge \ell^i_{\langle m,n \rangle} \in s \vee \ell^j_{\langle p,q \rangle} \in s \wedge ((m \leqslant n) \vee (p \leqslant q) \vee (n < p)) \Big\}.$$

5.1 Query Optimization

We next discuss how we can process the query graph.

Graph Partitions. We partition the graph $\mathcal{G}$ into a set of subgraphs $\mathcal{S}$ such that each subgraph $S \in \mathcal{S}$ either consists of a single vertex or a vertex pair (u, v) where vertex v is reachable from u. That way, each subgraph corresponds to an indexing unit for which we can retrieve its corresponding posting list from the five different indexes described in Sec. 3. Concretely, for a subgraph where the vertex pair consists of a word sequence and annotation, we retrieve their results using the 2-STITCH index (e.g., in Fig. 3, subgraph $(1, 2)$). For a subgraph, where the vertex is an annotation, we can retrieve their results using the ANNOTATION indexes. For a vertex that contains the wildcard marker $\boxed{*}$, we can either directly lookup their posting lists from the SKIP-GRAM index or compute the resultant posting list using N-GRAM indexes. For a vertex that contains the stack operator, we can retrieve its posting list using the 2-FRAGMENT indexes.

Greedy Graph Partitioning and Optimization. In a graph partitioning, certain attribute combinations can be retrieved more quickly than others, e.g., (ORG $\to$ *acquired*) versus (ORG $\to$ MONEY). Since, there can exist multiple graph partitions, we opt for that one which contains subgraphs whose corresponding posting lists are shortest in the indexes. Thus, a naïve decomposition of the graph into subgraphs in which each vertex is adjacent to each other (e.g., $\mathcal{S} = \{(1, 2), (2, 3), (3, 4), (3, 5)\}$ in Fig. 3) may not correspond to an efficient query execution plan. To speedup the query processing, we partition the graph in a *greedy manner*. Specifically, we seek **anchor vertices**, that correspond to bindings for n-grams (e.g., trigrams) and bindings for annotation types (e.g., MONEY) whose document frequency is less than other bindings in $\mathcal{G}$. Therefore, a subgraph (indexing unit) whose single vertex comprises of an anchor vertex shall have overall document frequency less than the anchor vertex by itself. Concretely, we first identify anchor vertices whose document frequencies are least amongst the bindings in the query graph $\mathcal{G}$ using dictionaries. Second, we compute subgraphs of vertex pairs: one of which is an anchor vertex and the other is reachable from the anchor vertex. Consider the query in Fig. 3, where the anchor node corresponds to the bigram *invested in*. Using this anchor vertex, we can partition the query graph as $\mathcal{S} = \{(1, 2), (2, 3), (2, 4), (2, 5)\}$, where each vertex is reachable from the anchor vertex. **Direct Index** can be further used in conjunction with the inverted indexes to speed up the processing of the query graph. We do this by keeping track of the number of common documents when processing the subgraphs in the graph partitioning $\mathcal{S}$. When this number is small (e.g., $\leqslant 25$), we can switch over to the DIRECT index to inspect the annotation layers for the remaining subgraphs in the partition.

5.2 Assembling the Puzzle (Raw Table)

Using the posting lists for the subgraphs in the partitioned graph, we assemble the complete text regions as evidences. This is done by computing the overlaps of the positional spans for the text regions in each document, with respect to the anchor vertex. This process of assembling the text regions as evidences is illustrated in Fig. 3. The assembled text regions help us to generate the raw table $\overline{\mathcal{T}}$. The text regions gathered as evidences however contain only the positional spans. At this stage, we consider only those positional spans that are short and span a sentence. We prefer concise sentences as they yield semantically meaningful rows in the table. To do this, we first rank positional spans by increasing length. Then, we check that they lie within a sentence using the sentence boundaries stored in the DIRECT index. These positional spans are next filled in with values for the various bindings in the structured query $\mathcal{Q}$. To fill in the values, we turn to the DIRECT index that stores within it the values for the annotations and the word sequences corresponding to the assembled positional spans. To generate the raw table, we instantiate a table with the number of columns equal to the number of bindings present in the structured query $\mathcal{Q}$. For each retrieved text region, we create a row in the table. Then, for each binding we lookup its cell value using the DIRECT index. At this step, we additionally verify the multiplicity constraints, if present. Also, if no value for a binding could be found, we fill its corresponding cell value as NULL. The NULL values are inferred from other near-duplicate rows using LINK and ANALYSIS operators.

for a billion dollars	$\equiv$	$\$10^9$	$\equiv$	$[10^9, 10^9]$
for over a billion dollars	$\equiv$	$> \$10^9$	$\equiv$	$[10^9, 10^9 + \Delta]$
for under a billion dollars	$\equiv$	$< \$10^9$	$\equiv$	$[10^9 - \Delta, 10^9]$
for around a billion dollars	$\equiv$	$\sim \$10^9$	$\equiv$	$[10^9 - \Delta, 10^9 + \Delta]$

Figure 4: Modeling uncertainty in numerical expressions.

6 SEMANTIC LINK OPERATOR

LINK operators group together near-duplicate mentions of text, entities, temporal, and numerical values in the raw table to generate the final table. The LINK operators provide functionality that is similar to that of deduplication in databases [19, 27]. However there the focus has been on linking records using only surface forms of attribute values. In contrast, our LINK operators take into account the context (or document) from which the row has been derived and collection-level statistics. Furthermore, we model the semantics behind the annotations that are part of the table schema when applying LINK. We model two kinds of semantics: text and numerical (Sec. 6.1 and Sec. 6.2). We model the semantics of text for the annotation types of: part-of-speech and named entities of types PERSON, ORGANIZATION, LOCATION, and MISC. We model the semantics of numbers for the annotation types: DATE, TIME, MONEY, PERCENT, and NUMBER. Additionally, we can locally resolve NULL values (local NULL resolution) (Sec. 6.3) by using the provenance of each row.

6.1 Semantic Model for Text

To link word sequences that refer to the same entity (PERSON, ORGANIZATION, and LOCATION) or concept (MISC) in different rows we rely on three similarity computations: surface, contextual, and global.

Surface Similarity establishes similarity between two strings in cell values using traditional edit-distance based measures. To this end, we use the Jaro-Winkler similarity [29], to compute the similarity between two text cell values $\overline{c}_1$ and $\overline{c}_2$ containing text . We denote this surface level similarity measure by: $\text{sim}_{\texttt{surface}}(\overline{c}_1, \overline{c}_2)$. For example, $\text{sim}_{\texttt{surface}}(\textit{motorola}, \textit{motorola mobility}) = 0.91$.

Contextual Similarity computes the similarity between the originating text regions of the cell values. For example, we can link together the word sequences, *youtube* and *video sharing* based on the similarity of their contexts, e.g., *google acquired the video website youtube* and *google buys out video sharing platform, youtube*. Concretely, the local text similarity is defined below:

$$\underset{\texttt{context}}{\text{sim}}(\overline{c}_1, \overline{c}_2) = \frac{|\ \texttt{text}(\overline{c}_1) \cap \texttt{text}(\overline{c}_2)\ |}{|\ \texttt{text}(\overline{c}_1) \cup \texttt{text}(\overline{c}_2)\ |}. \quad (4)$$

Eq. 4 captures the Jaccard coefficient between the bag of words for the matched text regions, $\texttt{text}(\overline{c}_1)$ and $\texttt{text}(\overline{c}_2)$, that help derive the cell values, $\overline{c}_1$ and $\overline{c}_2$.

Global Similarity computes text similarity by leveraging co-occurrence statistics aggregated over the entire document collection. For instance, we can link the entities referred by the phrases, *youtube* and *video sharing* based on the co-occurrence counts of { *youtube*, *video* } and { *youtube*, *sharing* }. To compute this similarity we leverage the SKIP-GRAM dictionaries that contain the document frequencies (df) of word pairs $\{w_1, w_2\}$. This global text similarity is defined below:

$$\underset{\texttt{global}}{\text{sim}}(\overline{c}_1, \overline{c}_2) = \frac{1}{Z} \cdot \sum_{w_1 \in \text{words}(\overline{c}_1)} \sum_{w_2 \in \text{words}(\overline{c}_2)} \frac{\text{df}(\{w_1, w_2\})}{|D|}, \quad (5)$$

where, $Z = |\text{words}(\overline{c}_1)| \cdot |\text{words}(\overline{c}_2)|$ is a normalization constant. The equation above captures the global similarity by computing the co-occurrence frequency of words in the cell values $\overline{c}_1$ and $\overline{c}_2$. The complete text similarity between two cell values $\overline{c}_1$ and $\overline{c}_2$ can now be defined as:

$$\underset{\texttt{text}}{\text{sim}}(\overline{c}_1, \overline{c}_2) = \frac{1}{3}\left[\underset{\texttt{surface}}{\text{sim}}(\overline{c}_1, \overline{c}_2) + \underset{\texttt{context}}{\text{sim}}(\overline{c}_1, \overline{c}_2) + \underset{\texttt{global}}{\text{sim}}(\overline{c}_1, \overline{c}_2)\right]. \quad (6)$$

We make the above design choice primarily for scalability reasons. Our method leverages pre-computed word co-occurrence statistics, which avoids computing transformed text representations (e.g., for neural embedding methods) at query time, thus speeding up the similarity computation.

6.2 Semantic Model for Numbers

Numerical values in the form of mentions of money, percentages, date, and time can be very vague and uncertain. For instance, the numerical mention in Fig. 2 is disambiguated to $> \$1 \times 10^9$. This numerical expression can refer to an infinite number of uncertain intervals. Similarly, a temporal expression such as *the 60s* can refer to a multitude of time intervals. Therefore, it becomes essential that we model their uncertainty to compute similarity between numerical values when applying the LINK operator.

To incorporate uncertainty in numerical values, we model a numerical expression, whose values belong to a domain $\Sigma_{\mathcal{N}}$, by associating an interval with it: $[b, e]$, where b denotes the begin and e the end. A temporal expression (or date) can be converted to a numerical expression by representing the dates as UNIX epochs (i.e., the number of milliseconds passed since 1970-01-01). We model the uncertainty based on the annotation type (e.g., date or numerical) and its value. Fig. 4 shows how uncertainty in numerical expressions can be modeled. The uncertainty Δ for annotations of DATE and TIME is determined by the difference between two consecutive time elements at a given granularity (e.g., $\Delta =$ *1* year). The uncertainty Δ for the rest of the numerical annotations is equal to half of the value being modeled (e.g., for the PERCENT annotation value of *50%*, Δ is equal to *25%*). The similarity between two numerical cell values $\overline{c}_1$ and $\overline{c}_2$ is:

$$\underset{\texttt{number}}{\text{sim}}(\overline{c}_1, \overline{c}_2) = \frac{|\overline{c}_1 \cap \overline{c}_2|}{|\overline{c}_1 \cup \overline{c}_2|}. \quad (7)$$

The denominator in Eq. 7 represents the number of numerical values at a fixed granularity that can be referenced by the union of the interval representation for $\overline{c}_1$ and $\overline{c}_1$. The numerator computes the extent of agreement in values between $\overline{c}_1$ and $\overline{c}_2$.

6.3 Local Resolution of NULL Values

NULL values in the raw table arise if the $\boxed{?\ell}$ operator is used to relax the match for the bindings. Unlike traditional imputation techniques in databases [12, 20] and open-IE approaches, JIGSAW can leverage the provenance of a raw row to infer or estimate the NULL value. To resolve NULL values, we make a *narrative assumption*: the provenance for the row bearing the NULL value is contained in a document that describes a narrative of related events or concepts. For instance, an acquisition made by Google for an undisclosed amount may be described by a news article by comparing it to related acquisitions. To resolve the NULL value locally we describe three methods: scoping, proximity, and semantic redundancy.

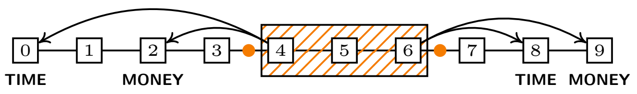


Figure 5: Inferring NULL values using the context surrounding the matched text region. Circular nodes represent sentence boundaries. Shaded region corresponds to the matched text region.

Scoping for resolving NULL values relies on frequency for named entity annotation type in the document containing the evidence. While, for numbers and time, the NULL value is resolved by constructing an interval using the minimum and maximum values of the same annotation type in the document containing the evidence. For instance, in Fig. 5 a NULL for annotation type of TIME can be estimated by constructing an interval using the annotation values present on position 0 and 8.

Proximity for resolving NULL values considers only nearby annotation values for estimation. This way, we can restrict ourselves to few (e.g., three) nearby values for resolving the NULL values. For example, in Fig. 5 we can resolve a NULL value for MONEY by looking at only the annotation at position 2 as the first nearest value.

Semantic Redundancy for resolving NULL values considers frequency in semantic models for text or numbers. Thus, for estimating the NULL values for named entity types for PERSON, ORG, and LOC we consider the semantic similarity measures discussed in Sec. 6.1. For estimating the NULL values for annotation types of NUMBERS and TIME, we consider the similarity measures described in Sec. 6.2.

The user can select based on the application domain from the above three methods for filling in the NULL values to yield the best table. For instance, for entity-centric queries, the proximity method, works well (in a manner similar to that of co-reference resolution). For event-centric queries, the semantic redundancy method is more suitable for local resolution of NULL values.

6.4 LINK$\left(\overline{\mathcal{T}}, \{a_1, a_2, \ldots, a_n\}, \theta\right)$ Operator

The operator LINK $(\overline{\mathcal{T}}, \{a_1, a_2, \ldots, a_n\}, \theta)$ takes as an input the raw table $\overline{\mathcal{T}}$; an attribute set $\{a_1, a_2, \ldots, a_n\}$ to link by; and a threshold θ to determine the degree of similarity between rows. The LINK operator outputs sets of rows that are near-duplicates.

Linking of rows in the raw table $\overline{\mathcal{T}}$ by attributes is done as follows. First, each row in the raw table $\overline{\mathcal{T}}$ is considered related to every other row. That is, we model the raw table as a complete undirected graph. Each undirected edge in the graph is weighted by a similarity value that is computed attribute-wise. That is, corresponding attributes from both rows are compared using Eq. 6 for text-based attributes and Eq. 7 for numerical attributes:

$$\mathrm{sim}(\overline{r}_1, \overline{r}_2) = \frac{1}{N} \cdot \sum_{a \in \{a_1, \ldots, a_k\}} \mathrm{sim}\left(\mathrm{value}(\overline{r}_1, a), \mathrm{value}(\overline{r}_2, a)\right), \quad (8)$$

where, the function value$(\overline{r}, a)$ returns the cell value in the row $\overline{r}$ for the attribute a and N denotes the total number of attributes (or columns in the table). Our model for a row in a table $\overline{r} \in \overline{\mathcal{T}}$ (see Eq. 2) allows for multiple cell values. To compute similarity between rows that contain multiple cell values for a single attribute, we compute their average pair-wise similarity:

$$\mathrm{sim}(\overline{r}_1, \overline{r}_2) = \frac{1}{N} \cdot \sum_{a \in \{a_1, \ldots, a_k\}} \frac{1}{Y} \cdot \sum_{\overline{c}_1 \in \mathrm{value}(\overline{r}_1, a)} \sum_{\overline{c}_2 \in \mathrm{value}(\overline{r}_2, a)} \mathrm{sim}\left(\overline{c}_1, \overline{c}_2\right),$$

Algorithm 1: FLAT and SCORE operators.

```
Input :Connected Component, $S = \{\overline{r}_1, \overline{r}_2, \ldots, \overline{r}_n\}$.
Function FLAT($S = \{\overline{r}_1, \overline{r}_2, \ldots, \overline{r}_n\}$)
    $r_{rep} \leftarrow \emptyset$ // Create a new representative row for S.
    for $a_i \in r_{rep}$ do
        value $\leftarrow \arg\max(\mathrm{sim}(c_i, \forall c_j \in$ S.values($a_i$) $\setminus c_i))$
        $r_{rep}$.put($a_i$, value)
    return $r_{rep}$
Input :The Final Table, $\mathcal{T} = \{r_1, r_2, \ldots, r_n\}$.
Function SCORE($\mathcal{T}$)
    // Compute support for each row in the final table.
    for $r \in \mathcal{T}$ do
        r.score = $^{\#\text{raw rows forming } r}/_{\#\text{total rows in raw table } \overline{\mathcal{T}}}$
    Sort($\mathcal{T}$) // Sort the rows in table by descending support.
    $\mathcal{T}_{new} \leftarrow \emptyset$
    while $\mathcal{T}$ is not empty do
        $\mathcal{T}_{new}$.append($\arg\max((1 - \mathrm{sim}(r_i, \forall r_j \in \mathcal{T}_{new} \setminus r_i)/|\mathcal{T}|)))$
        remove the row appended to $\mathcal{T}_{new}$ from $\mathcal{T}$
    return $\mathcal{T}_{new}$
```

where, $Y = |\mathrm{value}(\overline{r}_1, a)| \cdot |\mathrm{value}(\overline{r}_2, a)|$ is a normalization factor. Note that, the similarity of a cell value to a NULL value, that could not be resolved using local context, is defined to be zero:

$$\mathrm{sim}(\mathrm{NULL}, c) = 0. \quad (9)$$

Second, we find connected components in the weighted undirected graph representing $\overline{\mathcal{T}}$. A subgraph is considered connected if each edge in it has an edge weight greater than or equal to a threshold θ. Put another way, connected components can be found by removing all the edges in weighted graph that have weight less than the threshold θ. The remaining subgraphs are then clusters of related rows with respect to their attribute-wise similarity.

7 ANALYSIS OPERATORS

The LINK operator groups together near-duplicate raw rows using semantics of text and numbers. The ANALYSIS operators work in conjunction with LINK operators to flatten the group of rows into a representative row and to assign a score to each representative row for ranking. Additionally, when flattening a group of rows we can infer NULL values, those which could not be resolved locally, from other near-duplicate rows (global NULL resolution). We next describe these three operators: FLAT, SCORE, and RANK.

FLAT$(S = \{\overline{r}_1, \overline{r}_2, \ldots, \overline{r}_k\})$ Operator. The final table $\mathcal{T}$ consists only of representative rows $r \in \mathcal{T}$ derived from each connected component $\{\overline{r}_1, \overline{r}_2, \ldots, \overline{r}_k\} \in \overline{\mathcal{T}}$ discovered by LINK. Each representative row $r \in \mathcal{T}$, consists of cell values from different rows in the connected component. The selection of the cell values for the representative row is done by computing the similarity of a cell value from a row to all the other cell values for that attribute in the set S. The cell value that is most similar to the others is chosen to be part of the representative row. In this step, the representative row can infer the value for NULLs, that could not be resolved using local context, from other rows' cell values; we refer to this as **global resolution of NULL values**. The global NULL resolution thus resolves NULLs using cross-document evidences. We make the above design choice primarily to resolve NULL values independently from other attributes. Alternative design choices (e.g., selecting the most similar row in its entirety from the group) are less helpful in global NULL resolution. The FLAT operator is described in Algorithm 1.

Table 1: Annotated document collection size and statistics.

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
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: Index sizes in Gigabytes (GB).

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

Table 3: Query testbed statistics.

CATEGORY	#ENTITIES	PREDICATE	BINDINGS	EXAMPLE ENTITY
OLYMPIANS	265	PARTICIPANT OF	!LOCATION, ?TIME	*Usain Bolt*
MARRIAGE	237	SPOUSE	!PERSON , ?TIME	*Bob Dylan*
FOOTBALLERS	101	SPORTS TEAM	!ORG , ?TIME	*Kaká*
CEOS	87	CEO	!PERSON , ?TIME	*Google*
ACQUISITIONS	58	ACQUIRE	!ORG , ?TIME, ?MONEY	*Takeda*

$\mathtt{SCORE}(S = \{\bar{r}_1, \bar{r}_2, \dots, \bar{r}_k\})$ **Operator.** For each connected component $S \in \overline{\mathcal{T}}$ (or $r \in \mathcal{T}$), the SCORE operator assigns a value indicating: the size of S and the novelty of S amongst other connected components in $\overline{\mathcal{T}}$:

$$\mathtt{score}(r \in \mathcal{T}) = \text{support}(r \in \mathcal{T}) \cdot \text{diversify}(r \in \mathcal{T}),$$

where, $\text{support}(r \in \mathcal{T}) \equiv \text{support}(S \in \overline{\mathcal{T}}) = |S|/|\overline{\mathcal{T}}|$. To diversify the representative rows in the final table $\mathcal{T}$, the objective is to order rows in the final table such that a row is highly dissimilar to the rows above it. The SCORE operator is also described in Algorithm 1.

RANK **Operator**. We can order the rows in the final table by two methods. First, we can simply rank $r \in \mathcal{T}$ by the scores generated by the SCORE operator: $\text{rank}_{\mathtt{score}}$. Second, we can rank by the average length of the annotated text regions (text) supporting the row $r \in \mathcal{T}$. Rank by length can be defined as the average inverse length of the annotated text region span:

$$\underset{\mathtt{length}}{\text{rank}}\,(r \in \mathcal{T}) \equiv \underset{\mathtt{length}}{\text{rank}}\,(S \in \overline{\mathcal{T}}) = \frac{1}{|S|} \sum_{\bar{r} \in S} \frac{1}{|\,\mathtt{text}(\bar{r})\,|}.$$

Thus, each row in the final table $r \in \mathcal{T}$, contains rows that have been pieced together from partial, redundant, and paraphrased information from the text regions obtained for a structured query $\mathcal{Q}$ from large annotated document collection.

8 EVALUATION

In this section, we describe the evaluation setup of our experiments.

Annotated Document Collections and their Indexes. To evaluate JIGSAW, we considered three large news archives. The New York Times annotated corpus consists of around two million articles published during 1997-2007 [8]. The fifth edition of the English Gigaword consists of around ten million articles from seven different sources published during 1995-2010 [9]. GDelt news archive comprises of around fourteen million articles related to events available at the GDelt project website [3]. All three document collections, were preprocessed with the Stanford CoreNLP toolkit to obtain annotation for part-of-speech, named entity, temporal expressions, and numerical values. Statistics for the annotated document collections are displayed in Table 1. For each collection, we created the five indexes and the direct index (see Sec. 3). We store our indexes using HBase, a distributed and extensible record store. We list the indexes and their sizes for each document collection in Table 2.

Jigsaw Puzzles (Query Testbed). JIGSAW can provide multiple answers in the form of a table for a query where many answers are correct (e.g., *google acquisitions*). To measure the quality of the generated table, we need to identify queries for which we need to link text, time, or numbers. To this end, we constructed a testbed of 748 tabular queries concerning acquisitions, CEOs, Olympians, footballers, and marriages for popular entities. We obtained companies and their acquisitions from CrunchBase [1]. These prominent 58 organizations were identified using Fortune 500 [2], large software [7] and manufacturing [6] companies lists. For the remaining categories, we constructed the tables from the Wikidata Knowledge Graph. Specifically to generate these queries, we restrict ourselves to prominent named entities in Wikidata. Where, the prominence of an entity in Wikidata is determined using the concept of *sitelinks* [10]. To instantiate the structured queries, we use the following query template: $\langle$ENTITY, PREDICATE, UNORDERED BINDINGS$\rangle$. The template is then filled with aliases of entities (e.g., [*google* | *search giant*]), paraphrases for the predicates (e.g., [*acquired* | *takeover*]) from Wikidata and bindings for the requisite category (e.g., ORG, TIME, and MONEY for acquisitions — see Table 3). For example, a query for ACQUISITIONS (see Table 3) is shown below:

$$\Big\langle ![\textit{google}|\textit{google inc.}|\textit{google llc}], ![\textit{acquires}|\textit{acquired}|\textit{acquisition}| \textit{takeover}|\textit{bought}|\textit{buys}|\textit{scoops up}|\textit{to buy}|\textit{slurps}], \{!\text{ORG}, ?\text{TIME}, ?\text{MONEY}\}\Big\rangle.$$

Quality of the Generated Tables is measured by comparing against the ground truth extracted from Crunchbase and Wikidata. That is, for each row in the ground truth table $r_{\text{true}} \in \mathcal{T}_{\text{true}}$, we seek its equivalent row in the generated table and compare them in their respective semantic models of representation. We can therefore establish the notions of precision and recall using the semantic similarity measures discussed in Sec. 6.

Precision between the generated table $\mathcal{T}$ and the ground truth table $\mathcal{T}_{\text{true}}$ is computed by checking that each row in the generated table $r \in \mathcal{T}$ finds a corresponding equivalent row in the ground truth table $r_{\text{true}} \in \mathcal{T}_{\text{true}}$. This can be written as:

$$\text{precision}(\mathcal{T}, \mathcal{T}_{\text{true}}) = \frac{1}{|\mathcal{T}|} \sum_{r \in \mathcal{T}} \underset{r_{\text{true}} \in \mathcal{T}_{\text{true}}}{\arg\max} \; \text{sim}(r, r_{\text{true}}), \tag{10}$$

where, the similarity between two rows, $\text{sim}(r, r_{\text{true}})$, is computed using Eq. 8 described in Sec. 6.4. For text based attributes, we seek the maximum similarity match between the generated cell attribute and possible aliases in the ground truth.

Recall between the generated table $\mathcal{T}$ and the ground truth table $\mathcal{T}_{\text{true}}$ measures how many rows from the ground truth table are found in the generated table. This can be stated as:

$$\text{recall}(\mathcal{T}, \mathcal{T}_{\text{true}}) = \frac{1}{|\mathcal{T}_{\text{true}}|} \sum_{r_{\text{true}} \in \mathcal{T}_{\text{true}}} \underset{r \in \mathcal{T}}{\arg\max} \; \text{sim}(r, r_{\text{true}}). \tag{11}$$

8.1 Setup

Currently, there exists no system that can generate tables for user-defined schema over annotated document collections. To evaluate JIGSAW, we consider alternative design choices using various building blocks in our system.

Baseline Basic resolves NULL values for temporal and numerical expressions using *scoping*. Linking of rows, relies on surface level similarity and redundancy. Flattening of connected components is done by considering the most frequent textual cell value; while for numbers and time, minimum and maximum of the cell values in the group is considered. For ranking, only support is considered.

Baseline Advanced resolves NULL values for temporal and numerical expressions using *proximity*. Baseline Advanced links rows based on text using surface and contextual similarity as discussed in Sec. 6.1. Whereas, linking for numerical expressions is based on simple interval overlaps. Flattening of connected components is done by considering the most frequent textual cell value, while for numbers and time the most frequent interval is chosen as representative. For ranking, here also only support is considered.

Systems JIGSAW and JIGSAW++, are our proposed systems. JIGSAW applies the semantic redundancy method for local NULL value resolution. JIGSAW++ considers the semantic redundancy method for event-centric queries (i.e., acquisitions and Olympians) and proximity method for entity-centric queries (i.e., CEOs, footballers, and marriages). For flattening a group of rows, JIGSAW considers semantic redundancy and relatedness for numerical values and frequency for text attributes. JIGSAW++ considers the same method for event-centric queries. For entity-centric queries JIGSAW++ considers the frequency based method for both text and numerical attributes. For ranking, both support and diversity are considered.

Significance Tests results between the baseline Basic and JIGSAW are shown by $\triangle$. Statistically significant results between baseline Advanced and JIGSAW are marked by $\blacktriangle$. The significance was computed using the two-tailed paired t-Test at $\alpha = 0.05$.

Hardware. The storage for the indexes is a cluster of twenty machines running Cloudera CDH 5.90 version of Hadoop and HBase. Each machine in the Hadoop cluster is equipped with up to a 24 core Intel Xeon CPU with 3.50 GHz processing speed, up to 128 GB of RAM, and up to eight 4 TB worth of secondary storage. We perform all evaluations on a high-memory compute node, with 96 core Intel Xeon CPU at 2.66 GHz processing speed and 1.48 TB of RAM.

8.2 Results

We executed the baselines and systems for all of the queries in each query category in Table 3. We considered three different values of similarity thresholds $\theta \in \{0.25, 0.50, 0.75\}$ for the LINK operator, to determine its best value. We then computed precision, recall, and F_1 (harmonic mean of precision and recall) at values of top-$k \in \{10, 25, 50\}$. The results for precision and recall averaged over all query categories for each collection are shown in Table 4 and 5. We summarize the F_1 values over all collections in Table 6. The best value of the measures is highlighted for the corresponding threshold value. For the collections NYT, Gigaword, and GDelt we were able to generate 344, 444, and 414 tables respectively. The number of tables vary per collection due to varying time periods of their reporting. For example, NYT does not contain any acquisitions for Twitter whereas Gigaword and GDelt do.

Table 4: Precision over all categories in the testbed.

	TOP-k	10			25			50		
	θ	0.25	0.50	0.75	0.25	0.50	0.75	0.25	0.50	0.75
NYT	BASIC	0.04	0.04	0.23	0.02	0.02	0.17	0.01	0.01	0.12
	ADVANCED	0.04	0.17	0.23	0.01	0.10	0.18	0.01	0.06	0.14
	JIGSAW	$0.05^{\triangle}_{\blacktriangle}$	$0.22^{\triangle}_{\blacktriangle}$	0.24	$0.02_{\blacktriangle}$	$0.16^{\triangle}_{\blacktriangle}$	$0.20^{\triangle}_{\blacktriangle}$	0.01	$0.12^{\triangle}_{\blacktriangle}$	$0.17^{\triangle}_{\blacktriangle}$
	JIGSAW++	$0.05^{\triangle}_{\blacktriangle}$	$0.22^{\triangle}_{\blacktriangle}$	0.25	$0.02_{\blacktriangle}$	$0.17^{\triangle}_{\blacktriangle}$	$0.21^{\triangle}_{\blacktriangle}$	0.01	$0.13^{\triangle}_{\blacktriangle}$	$0.18^{\triangle}_{\blacktriangle}$
	TOP-k	10			25			50		
	θ	0.25	0.50	0.75	0.25	0.50	0.75	0.25	0.50	0.75
GIGAWORD	BASIC	0.04	0.04	0.27	0.01	0.02	0.23	0.01	0.01	0.18
	ADVANCED	0.04	0.22	0.29	0.02	0.15	0.25	0.01	0.09	0.21
	JIGSAW	$0.05^{\triangle}_{\blacktriangle}$	$0.27^{\triangle}_{\blacktriangle}$	0.26	$0.02^{\triangle}$	$0.23^{\triangle}_{\blacktriangle}$	0.25	0.01	$0.20^{\triangle}_{\blacktriangle}$	$0.23^{\triangle}_{\blacktriangle}$
	JIGSAW++	$0.05^{\triangle}_{\blacktriangle}$	$0.28^{\triangle}_{\blacktriangle}$	0.27	$0.02^{\triangle}$	$0.24^{\triangle}_{\blacktriangle}$	0.25	0.01	$0.20^{\triangle}_{\blacktriangle}$	$0.24^{\triangle}_{\blacktriangle}$
	TOP-k	10			25			50		
	θ	0.25	0.50	0.75	0.25	0.50	0.75	0.25	0.50	0.75
GDELT	BASIC	0.04	0.05	0.27	0.02	0.02	0.19	0.01	0.01	0.15
	ADVANCED	0.04	0.17	0.27	0.02	0.12	0.21	0.01	0.08	0.17
	JIGSAW	$0.06^{\triangle}_{\blacktriangle}$	$0.24^{\triangle}_{\blacktriangle}$	$0.30^{\triangle}_{\blacktriangle}$	0.02	$0.19^{\triangle}_{\blacktriangle}$	$0.26^{\triangle}_{\blacktriangle}$	0.01	$0.15^{\triangle}_{\blacktriangle}$	$0.24^{\triangle}_{\blacktriangle}$
	JIGSAW++	$0.06^{\triangle}_{\blacktriangle}$	$0.25^{\triangle}_{\blacktriangle}$	$0.31^{\triangle}_{\blacktriangle}$	0.02	$0.20^{\triangle}_{\blacktriangle}$	$0.27^{\triangle}_{\blacktriangle}$	0.01	$0.15^{\triangle}_{\blacktriangle}$	$0.25^{\triangle}_{\blacktriangle}$

Table 5: Recall over all categories in the testbed.

	TOP-k	10			25			50		
	θ	0.25	0.50	0.75	0.25	0.50	0.75	0.25	0.50	0.75
NYT	BASIC	0.31	0.32	0.38	0.31	0.32	0.39	0.31	0.32	0.40
	ADVANCED	0.29	0.35	0.36	0.29	0.36	0.38	0.29	0.36	0.39
	JIGSAW	0.29	$0.35^{\triangle}$	0.33	0.29	$0.36^{\triangle}$	0.35	0.29	$0.37^{\triangle}_{\blacktriangle}$	0.35
	JIGSAW++	$0.30_{\blacktriangle}$	$0.36^{\triangle}_{\blacktriangle}$	0.36	$0.30_{\blacktriangle}$	$0.38^{\triangle}_{\blacktriangle}$	0.38	$0.30_{\blacktriangle}$	$0.39^{\triangle}_{\blacktriangle}$	0.39
	TOP-k	10			25			50		
	θ	0.25	0.50	0.75	0.25	0.50	0.75	0.25	0.50	0.75
GIGAWORD	BASIC	0.29	0.30	0.40	0.29	0.30	0.43	0.29	0.30	0.44
	ADVANCED	0.29	0.37	0.39	0.29	0.38	0.42	0.29	0.38	0.44
	JIGSAW	0.29	$0.37^{\triangle}$	0.35	0.29	$0.40^{\triangle}_{\blacktriangle}$	0.37	0.29	$0.42^{\triangle}_{\blacktriangle}$	0.39
	JIGSAW++	$0.30_{\blacktriangle}$	$0.39^{\triangle}_{\blacktriangle}$	0.37	$0.30_{\blacktriangle}$	$0.42^{\triangle}_{\blacktriangle}$	0.41	$0.30^{\triangle}_{\blacktriangle}$	$0.44^{\triangle}_{\blacktriangle}$	0.43
	TOP-k	10			25			50		
	θ	0.25	0.50	0.75	0.25	0.50	0.75	0.25	0.50	0.75
GDELT	BASIC	0.32	0.32	0.40	0.32	0.32	0.42	0.32	0.32	0.43
	ADVANCED	0.32	0.37	0.38	0.32	0.39	0.41	0.32	0.39	0.43
	JIGSAW	0.31	$0.38^{\triangle}$	0.37	0.31	$0.40^{\triangle}$	0.39	0.31	$0.42^{\triangle}_{\blacktriangle}$	0.41
	JIGSAW++	0.32	$0.39^{\triangle}_{\blacktriangle}$	0.40	0.32	$0.42^{\triangle}_{\blacktriangle}$	0.43	0.32	$0.44^{\triangle}_{\blacktriangle}$	0.44

Table 6: F_1 for all testbed categories and all collections.

	TOP-k	10			25			50		
	θ	0.25	0.50	0.75	0.25	0.50	0.75	0.25	0.50	0.75
OVERALL	BASIC	0.07	0.08	0.30	0.03	0.03	0.25	0.02	0.02	0.21
	ADVANCED	0.07	0.24	0.30	0.03	0.17	0.26	0.01	0.12	0.23
	JIGSAW	$0.09^{\triangle}_{\blacktriangle}$	$0.28^{\triangle}_{\blacktriangle}$	0.30	$0.04^{\triangle}_{\blacktriangle}$	$0.24^{\triangle}_{\blacktriangle}$	$0.28^{\triangle}_{\blacktriangle}$	$0.02_{\blacktriangle}$	$0.21^{\triangle}_{\blacktriangle}$	$0.27^{\triangle}_{\blacktriangle}$
	JIGSAW++	$0.09^{\triangle}_{\blacktriangle}$	$0.29^{\triangle}_{\blacktriangle}$	$0.31^{\triangle}$	$0.04^{\triangle}_{\blacktriangle}$	$0.25^{\triangle}_{\blacktriangle}$	$0.29^{\triangle}_{\blacktriangle}$	$0.02_{\blacktriangle}$	$0.22^{\triangle}_{\blacktriangle}$	$0.28^{\triangle}_{\blacktriangle}$

Effectiveness Results. In terms of precision (see Table 4), we observe that overall JIGSAW achieves the best performance. In terms of recall (see Table 5), we observe that JIGSAW performs at par or better when compared to the baselines. Note that perfect recall is not possible due to: temporal coverage of collections and KG incompleteness. When considering the F_1 score (see Table 6), JIGSAW provides a balanced performance of high precision and good recall as compared to the baselines that excel only in recall. The good performance of JIGSAW can be attributed to three key system design choices. First, the baselines are more sensitive to θ as they rely only on surface (Baseline Basic) or contextual (Baseline Advanced) similarities for text. Whereas, JIGSAW leverages surface, contextual, and global text similarities to link rows. Second, JIGSAW uses semantic redundancy for NULL value resolution that provides higher precision. However, the NULL resolution techniques utilized by the baselines produce less reliable estimates for temporal and numerical attributes. Third and finally, the techniques adopted by the baselines for flattening linked rows result in broader temporal and numerical representations that are less precise. In addition to this, JIGSAW is more robust and achieves higher precision at different similarity thresholds θ.

Efficiency Results. We evaluated JIGSAW for efficiency by executing a sample of 100 queries from the query testbed three times in cold-cache setting for each collection. To simulate cold caches, we shuffle the queries in between rounds. To contrast the performance of our query-driven system with existing open-IE systems: we measure the time needed to scan the entire collection on our Hadoop cluster once. This thus simulates the minimum amount of time an open-IE system shall take to just retrieve all the sentences for a query in an embarrassingly parallel manner. The results for the scan baseline are shown in Table 7. The results for the end-to-end run-times for our system JIGSAW and the baselines Basic and Advanced are shown in Table 8. From Table 7 we see that scanning the entire collection for each query is in the order of minutes. From Table 8, we see that end-to-end run times for generating tables using the baselines or our system JIGSAW are significantly less than a simple scan. From Table 8, we observe that JIGSAW takes more time to generate tables than the baselines. This is because the baselines only leverage surface-level and contextual similarities that are quick to compute as they only require in-memory operations. On the other hand, JIGSAW additionally leverages global similarity, that relies on lookups from the dictionaries. This additional time however provides us improved linking of raw rows. Overall, we see that we gain at least an order of 13.96× speedup and at most an order of 44.95× speedup over a simple scan of the collections.

9 RELATED WORK

Annotated document collections have been leveraged by several studies [15, 16, 22] for mining valuable data. [15] was seminal work in proposing key operators to analyze annotated text corpora using relational databases. [16, 22] describe efficient algorithms to perform search in tagged text corpora using inverted index operations. However, none of the above systems support table generation capabilities that can aggregate redundant, partial, and paraphrased text evidences. A recent survey on the use of web tables [13], describes the impact web tables have had on commercial search engines. The authors also describe progress that has been made in terms of augmenting web tables from additional data sources such as KGs. [14] aims to link predicates from KGs to relations between attributes in web tables in order to understand their schema. [30, 32] generate tables from KGs to answer keyword queries. The above approaches focus on leveraging existing structured resources (e.g., web tables and KGs) but not to generate tables from unstructured text. Knowledge graph population techniques rely on identifying salient extractions from documents or the Web [11]. Key works in this direction are [18, 25, 26, 28, 31]. [26] relies on distributed itemset mining for determining salient triples to be added to KGs. [28, 31] uses Markov logic to reconcile and canonicalize triples. [18] verifies the correctness of the extracted triples by using a combination of prior-knowledge learned using random-walks and neural-networks. [25] uses a suite of machine learning methods including embedding-based methods to verify the quality of triples to be added to its KG. However, all these methods are bound to a fixed schema for extraction. Furthermore, the discussed methods solely rely on offline methods of pre-computing the KG. JIGSAW, on the other hand, allows table generation for user-defined schema efficiently and interactively in a query-driven manner.

Table 7: Time in seconds taken to scan a collection on our cluster.

NYT	GIGAWORD	GDELT
111.00	396.00	604.00

Table 8: Run-times in seconds for our system JIGSAW.

COLLECTION	BASIC	ADVANCED	JIGSAW	JIGSAW++
NYT	3.68 ± 6.16	3.86 ± 6.85	7.63 ± 15.63	7.95 ± 16.28
GIGAWORD	8.81 ± 11.12	9.09 ± 10.84	16.30 ± 20.85	16.45 ± 22.09
GDELT	17.33 ± 35.15	17.90 ± 36.22	28.37 ± 43.48	27.99 ± 42.56

10 CONCLUSION

We presented JIGSAW, a system that generates tables from unstructured text for user-defined schema. To do so, we described `QUERY` operators to define the table schema. To speedup query processing we described a greedy query optimizer. To generate high-quality tables, we described `LINK` and `ANALYSIS` operators that leverage semantic models for text and numbers to group together near-duplicate rows. Our evaluation demonstrates that JIGSAW can generate high-quality tables from over 25 million documents efficiently.

REFERENCES

[1] Crunchbase. `https://www.crunchbase.com/`.
[2] Fortune. `http://fortune.com/fortune500/`.
[3] The GDELT Project. `https://www.gdeltproject.org/`.
[4] Google Tables. `https://research.google.com/tables`.
[5] Structured Snippets in Google Web Search. `https://ai.googleblog.com/2014/09/introducing-structured-snippets-now.html`.
[6] Largest Manufacturing Companies by Revenue . `https://en.wikipedia.org/wiki/List_of_largest_manufacturing_companies_by_revenue`.
[7] Largest Software Companies. `https://en.wikipedia.org/wiki/List_of_the_largest_software_companies`.
[8] NYT Annotated Corpus. `https://catalog.ldc.upenn.edu/LDC2008T19`.
[9] Gigaword Fifth Edition. `https://catalog.ldc.upenn.edu/LDC2011T07`.
[10] Wikidata: Sitelinks. `https://www.wikidata.org/wiki/Help:Sitelinks`.
[11] M. Banko et al. 2007. Open Information Extraction from the Web. *IJCAI 2007*.
[12] C. Batini et al. Methodologies for Data Quality Assessment and Improvement. *ACM Comput. Surv.* 41, 3 (2009), 16:1–16:52.
[13] M. J. Cafarella et al. Ten Years of WebTables. *PVLDB* 11, 12 (2018), 2140–2149.
[14] M. Cannaviccio et al. Towards Annotating Relational Data on the Web with Language Models. *WWW 2018*.
[15] L. Chiticariu et al. SystemT: An Algebraic Approach to Declarative Information Extraction. *ACL 2010*.
[16] C. L. A. Clarke et al. An Algebra for Structured Text Search and a Framework for its Implementation. *Comput. J.* 38, 1 (1995), 43–56.
[17] D. Crow. Google Squared: Web Scale, Open Domain Information Extraction and Presentation. *ECIR 2010 Industry Day*.
[18] X. Dong et al. Knowledge Vault: A Web-Scale Approach to Probabilistic Knowledge Fusion. *KDD 2014*.
[19] A. K. Elmagarmid et al. Duplicate Record Detection: A Survey. *IEEE Trans. Knowl. Data Eng.* 19, 1 (2007), 1–16.
[20] I. P. Fellegi and D. Holt. A Systematic Approach to Automatic Edit and Imputation. *J. Amer. Statist. Assoc.* 71, 353 (1976), 17–35.
[21] J. Gray et al. *The Data Journalism Handbook*.
[22] D. Gupta and K. Berberich. GYANI: An Indexing Infrastructure for Knowledge-Centric Tasks. *CIKM 2018*.
[23] J.W. Henry. 2013. Providing Knowledge Panels with Search Results. US Patent App. 13/566,489.
[24] P. J. Hong et al. 2018. Related entities. US Patent App. 15/798,175.
[25] T. M. Mitchell et al. Never-Ending Learning. *Commun. ACM* 61, 5 (2018), 103–115.
[26] N. Nakashole et al. Scalable Knowledge Harvesting with High Precision and High Recall. *WSDM 2011*.
[27] F. Naumann and M. Herschel. *An Introduction to Duplicate Detection.* Morgan & Claypool Publishers.
[28] Feng Niu, Ce Zhang, C. Ré, and J. W. Shavlik. Elementary: Large-Scale Knowledge-Base Construction via Machine Learning and Statistical Inference. *Int. J. Semantic Web Inf. Syst.* 8, 3 (2012), 42–73.
[29] W. E Winkler. String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage. (1990).
[30] M. Yang et al. Finding Patterns in a Knowledge Base using Keywords to Compose Table Answers. *PVLDB* 7, 14 (2014), 1809–1820.
[31] C. Zhang et al. DeepDive: Declarative Knowledge Base Construction. *Commun. ACM* 60, 5 (2017), 93–102.
[32] S. Zhang and K. Balog. On-the-Fly Table Generation. *SIGIR 2018*.