KNITTIR: Syntactical Text Indexing for Analytics

Thanh-Hi Anthony Vu
thvu@ntnu.no
Norwegian University of Science and Technology
Trondheim, Norway

Dhruv Gupta
dhruv.gupta@ntnu.no
Norwegian University of Science and Technology
Trondheim, Norway

Abstract

Scalable text analytics requires retrieval of similar text regions spread across millions of documents. It also requires that we can reason about entities by categorizing them; contrasting them to other entities; and ranking them using time and numbers. We present KNITTIR that assists in such complex text analytical tasks. KNITTIR uses semantic annotations such as parts-of-speech, named entities, and their syntactical relationships to words in text. To simplify text analytics, KNITTIR uses a new search framework wherein a vertical partitioning of semantically annotated text is used. This allows users to aggregate and manipulate evidences to their queries from multiple text regions spread across millions of documents. To scale analytical queries, KNITTIR creates indexes using the syntactical modeling of annotated text. Our experiments over 22 million documents show that KNITTIR obtains speedups of up to 60× for performing similarity search and up to 69K× for reasoning tasks.

CCS Concepts

• **Information systems** → **Search engine indexing**.

Keywords

Analytics; Text Reasoning; Temporal and Numerical Reasoning

ACM Reference Format:
Thanh-Hi Anthony Vu and Dhruv Gupta. 2024. KNITTIR: Syntactical Text Indexing for Analytics. In *Proceedings of Joint Conference on Digital Libraries (JCDL '24).* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3677389.3702486

1 Introduction

Scholars in digital humanities, social scientists, and computational journalists require the capability to process large document collections for analytics [19, 35, 52]. Analytical tasks for these users involve: text (words, phrases, and sentences); named entities (persons, organizations, and locations); and temporal and numerical expressions [35]. A typical set of analytical tasks for these users can be summarized as follows: the retrieval of pseudo-relevant documents; probing for query-specific text regions; organizing them in categories; contrasting them to other entities; and finally performing aggregation for relevant statistics [52]. Such complex analytical tasks are often manual, repetitive, and dredging [52].

JCDL '24, December 16–20, 2024, Hong Kong, China

ACM ISBN 979-8-4007-1093-3/24/12
https://doi.org/10.1145/3677389.3702486

Current search systems, however, are deemed insufficient by scholars in humanities for aforementioned complex analytical tasks [29, 30, 34, 52]. Harris et al. [29, 30] describe that search systems that rely on term-based indexes fail to provide similarity search capability, which is essential for scholars in humanities, as they rely on queries composed of *motifs* or templates to extract similar text regions. Modern vector-based indexes can provide similarity search capabilities; however, they lack explainability and manipulability. In particular, manipulability is important as Terras et al. [52] observed that scholars in digital humanities often require a handful of key queries to formulate their research problem that have *very slow* response times. An example analytics task they describe is reasoning about entities by: extracting entities from documents, computing their frequency over time, and contrasting them with other related entities over the same time period. If such capabilities can be provided as operators in search systems, much like native database operators, it would allow the scholars to perform complex text analytics tasks with ease. Moreover, a recent survey by Liu et al. [34] outlines that available text analysis platforms can not scale to large document collections. This is important as both Terras et al. [52] and Liu and Wang [35] mention that scholars in digital humanities search in an *iterative* manner, such that their queries can change based on the retrieved results. Therefore, highlighting the need for a search system that can execute analytical queries quickly.

In summary, the key desiderata for a search system to speedup such complex analytical tasks are: fast response times for similarity search; reasoning abilities as operators for categorizing entities, contrasting entities, and ranking them using temporal and numerical expressions. To serve such text-centric analytical needs of applied scientists, currently, there are two kinds of solutions available for text retrieval: term-based inverted indexes and vector-based indexes. We next describe the challenges with search systems that leverage term-based and vector-based indexes. We then outline how KNITTIR overcomes these challenges and simplifies complex text analytical tasks.

Term-based Indexes (e.g., Lucene [2, 6]) provide Boolean operators to retrieve pseudo-relevant documents. Key benefits of term indexes are: small indexes (comparable to raw collection size) and fast query response times. However, a key drawback of term indexes is that they lack the capability of performing similarity search directly. Indirect similarity search can be done using relevance feedback to improve the initial result set. Extensions to term indexes can provide proximity search (using word positions) and semantic search (using annotations). However, current implementations that index annotations [13, 26] draw them from paraphrase dictionaries or knowledge graphs (KGs). This limits contextual similarity search for domain-specific text (e.g., `green energy` is similar to `clean energy`) which can only be done using the collection.

Vector-based Indexes achieve similarity search, by transforming text into a high-dimensional representation such that each term in text is represented as a vector. Since, these representations are very sparse for large document collections, a more meaningful projected dense representation can be obtained via deep learning methods (e.g., word2vec [38]). For retrieval of similar text regions corresponding to a query essentially boils down to efficient computation of distances to determine nearest neighbors in the new projected dense space. This computation corresponds to finding vectors that have the maximum dot product with respect to the query vector [4, 25, 39]. Key advantages of vector-based indexes is that they can leverage several machine-level optimizations such as vector-based CPU and GPU instructions to speed up the result computation. Furthermore, they offer similarity search implicitly such that these aspects need not be explicitly modeled. This is particularly advantageous, for instance, when compared with pseudo-relevance feedback. However, there are several key disadvantages that prohibit their deployment for analytical tasks. First, the semantics are modeled implicitly by modeling text in a high-dimensional space and utilizing a projected dense representation. This allows the user little or no text manipulability. The user has control only over a distance computation between the query and the text vector. To provide any context or semantics the user has do multiple cycles of post-hoc analysis to arrive at the certainty of the results; thus, slowing down the overall analytics workflow. Moreover, for many of the compute-affordable dense representations it is impossible to recover contextual semantics as a bag-of-words text model is leveraged [38]. Second, to speed up the dot product computation several lossy compression and quantization techniques are applied such that results are never exact thus hurting high-recall analytic-centric tasks. In fact, vector-based indexes achieve complete recall only by using a linear scan [12]. Third, users are severely limited in utilizing vector-based indexes if they are compute-restrained. That is, if only commodity based hardware is available then there is little benefit of using such storage schemes. Fourth and finally, it has been well-documented that dense retrieval approaches do not offer any temporal and numerical reasoning capability [44] which is central to many analytical tasks.

Contributions. We present KNITTIR which overcomes aforementioned challenges. First, KNITTIR models text using its syntactical structure and semantic annotations for similarity search. Syntactical modeling provides users explainability of KNITTIR's search method versus the opaqueness of existing vector-based search [4, 25, 39]. Moreover, syntactical modeling provides domain-specific contextualization, which is not possible with dictionary and KG based indexing methods [13, 55]. Second, KNITTIR provides categorical, contrastive, temporal, and numerical reasoning for entities. Third and finally, KNITTIR streamlines analytical tasks with a new search framework of vertical partitioned annotated text regions.

An Example Analytics Workflow that can be accomplished with KNITTIR is shown in Figure 1. The user initiates the analytics workflow with the query *"williams wins olympic medal"* to retrieve similar text regions containing mentions of other athletes who have also won a medal or an award. Harris et al. [29, 30] describe that such search tasks are important for scholars in digital humanities to identify *parallel passages*. In Figure 1a, the results are presented as a vertically partitioned result set. Each row in the tabular representation contains text regions that are semantically, syntactically, and contextually similar to the input query. With this representation, the user can select horizontally a subset of rows to further analyze by reading the documents (using the associated metadata, i.e., document identifiers and publication dates) that contain the text region. Moreover, the user can select vertically a subset of entities which can then be further reasoned upon using additional context from the document collection. This is an important search task for scholars in digital humanities [52] and computational journalists [19]. Terras et al. [52] outline that an important task for scholars in humanities is to search for related entities and compare their relationship (e.g., using their frequency) over time (e.g., using document publication dates). Furthermore, Cohen et al. [19], emphasize that the ability to extract related entities in large document collections allows journalists to identify new *story patterns* and *narratives*. In Figures 1b-1g, we show how KNITTIR provides these capabilities through its query language that supports the ability to *categorize*, *contrast*, and *rank* (using temporal and numerical expressions) entities. In Figure 1b, the user applies the GROUP △ operator on *nadal* ⊕PERSON (where ⊕ associates the named entity annotation PERSON to *nadal*) to identify its related categories from the document collection. In Figure 1c, the user expands the category *athlete*⊕NN with additional entities using the EXPAND ▽

williams	wins	olympic_medal	TEXT REGION
benoit-samuelson ⊕ PERSON	win ⊕ VB	medal ⊕ MISC	[benoit-samuelson won the 1984 olympic gold medal]
williams ⊕ PERSON	win ⊕ VB	open ⊕ MISC	[williams won the australian open][williams won the australian open and the french open]
o'reilly ⊕ PERSON	win ⊕ VB	cup ⊕ MISC	[o'reilly 32 won three olympic gold medals and the 2015 world cup]
nadal ⊕ PERSON	win ⊕ VB	cup ⊕ MISC	[nadal has won the olympic gold medal in singles has won the davis cup]
williams ⊕ PERSON	clinch ⊕ VB	slam ⊕ MISC	[williams clinches a single-season grand slam]
williams ⊕ PERSON	earn ⊕ VB	globe ⊕ MISC	[williams earned a golden globe]

(a)

GROUP	TEXT REGION
champion ⊕NN	[nadal is the defending champion]
spaniard ⊕MISC	[spaniard nadal]
athlete⊕NN	[athletes like roger federer rafael nadal]

(b)

EXPAND	TEXT REGION
lebron_james ⊕PERSON	[athletes including lebron james]
serena_williams ⊕PERSON	[serena williams is the highest paid female athlete]
tiger_woods ⊕PERSON	[tiger woods is just the latest prima donna professional athlete]

(c)

OUTLIER	TEXT REGION
actor⊕NN	[actor but a terrible athlete]
student⊕NN	[athlete but a lousy student]
humanitarian ⊕JJ	[athlete but really a humanitarian]

(d)

COMPARE	TEXT REGION
nonathlete ⊕ NN	[athletes took more classes than nonathletes]
male ⊕ NN	[athletes get concussions at a higher rate than males]
man ⊕ NN	[athletes require more carbohydrates and protein than non-active men]

(e)

TIME	TEXT REGION
[2016-08-01, 2016-08-31]	[olympics 2016 will be held in august]
[2020-01-01, 2020-12-31]	[olympics will be held in 2020]
[2024-01-01, 2024-12-31]	[olympics come to paris in 2024]

(f)

NUMBER	TEXT REGION
[15.0,trophy]	[nadal increased his career haul to 15 grand slam trophies]
[<=2.0,point]	[nadal came within two points]
[%1.0-3.0, percent]	[nadal is getting further away from his goal of growing revenues 1-3 percent]

(g)

Figure 1: Text regions from the C4-News collection [43] given the query q ="*williams wins olympic medal*" for (a) similarity search ~ (q) (b) GROUP △(*nadal* ⊕PERSON) (c) EXPAND ▽(*athlete* ⊕NN) (d) OUTLIER ↯(*athlete* ⊕NN) (e) COMPARE ▷(*athlete* ⊕NN) (f) TIME T(*olympics* ⊕MISC) and (g) NUMBER N(*nadal* ⊕PERSON) operators.

operator. In Figure 1d, the user extracts phrases that do not usually belong to the category *athlete*⊕NN using the OUTLIER operator. In Figure 1e, the user compares the category *athlete*⊕NN with other related entities in the document collection using the COMPARE operator. In Figure 1f, the user extracts the temporal expressions associated with the entity *olympics*⊕MISC using the TIME operator. In Figure 1g, the user extracts numerical values along with their units related to the athlete *nadal*⊕PERSON using the NUMBER operator. With KNITTIR's query language, a user can compose analytics workflows in a manipulable manner that allows them to extract structured results at scale from millions of documents quickly.

2 KNITTIR

We now describe KNITTIR in detail.

2.1 Syntactical Text Model

Text analytics requires similarity search that can be achieved in three ways: bag-of-words, contextual, and syntactical modeling. We consider each of these design choices next.

Bag-of-Words (BoW) Text Model assumes independence between occurrence of terms thereby modeling a document as a multiset of terms. The BoW text model forms the basis for the vector-space model [36] and is the foundation of the term indexes [2, 6]. Similarity between query and documents is computed using the scalar dot product between their vector representations. Two key disadvantages of the BoW model are: contextual semantics are loosely captured and to retrieve exact text regions requires scanning documents. Approaches such as latent semantic indexing (LSI) [20] can be leveraged to obtain implicit categories over groups of terms for pseudo-relevant sets of documents. However, these approaches can not scale as they rely on expensive matrix factorization methods.

Context-based Text Model uses positional information in word sequences to determine its context. A word's context is modeled by a fixed-length window (i.e., n-grams or skip-grams). This text model is leveraged in term indexes [2, 6, 26, 28] for proximity and GREP-like search capability. Vector indexes also use this approach [38, 41] to obtain dense representations for text. A key limitation is that it models many word combinations which are not syntactically connected and thus not meaningful for semantic search. For example, in '*Williams and Phelps expected to win 2 and 4 medals, respectively*', the trigram '*2 and 4*' is not useful unless connected to the subjects of the sentence.

Syntax-based Text Model leverages parse trees obtained by natural language processing (NLP) tools to determine relationships (e.g., subject, object, or copula) between words in a sentence. Additionally, we can obtain annotations about roles the words play in a sentence via part-of-speech tags (e.g., nouns). Syntax-based text models have been used in many NLP tasks such as word sense disambiguation [53] and paraphrasing [14, 53] using contextual semantics. Syntax-based text models allow the capability to design operators that offer text compositionality and manipulability. Furthermore, syntax-based models allow us to keep only those combinations of terms in sentences that are syntactically related.

For text analytics, we need to enable similarity search to retrieve text regions that are syntactically, semantically, and contextually similar to queried word sequence. Furthermore, we need to

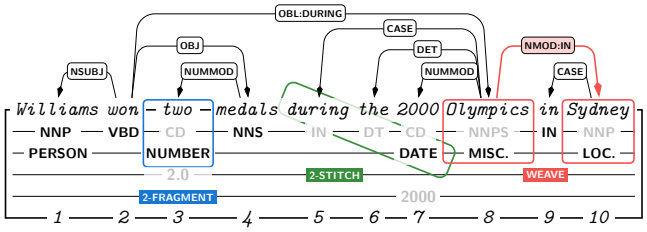

Figure 2: KNITTIR text model for index design and query processing.

enable reasoning capabilities to organize entities into categories; contrast entities; and rank entities using time and numbers. To do so, we design KNITTIR's text model that leverages the annotated text model [26] and the syntactical text model [14] for organizing annotations such as part-of-speech; named entities; resolved temporal and numerical expressions; and dependency parse trees. To describe KNITTIR's complete text model, we explain each of the aforementioned annotations.

Semantic Annotations. We model four types of annotations to model text semantics. First, we model part-of-speech annotations that convey the roles words play in the syntactical structure of the sentence (e.g., nouns, verbs, and adjectives). Second, we model coarse-grained named entity annotations that identify persons, organizations, locations, temporal and numerical expressions in a sentence. Third and fourth annotations, we model are resolved interval representations corresponding to the temporal and numerical expressions in text. Temporal and numerical expressions can be mentioned in an implicit manner (e.g., *tomorrow*) which further require their resolution with explicit mentions of time and numbers present as part of the document metadata (e.g., publication date) or other explicit mentions elsewhere in the document.

Syntactical Annotations. To model the text syntax, we use dependency parse trees. Unlike, constituency parse trees derived from context-free grammars, dependency parse trees do not provide any phrasal-structures [32]. Instead, dependency parse provides us with binary grammatical relationships between words in a sentence (e.g., nominal subject, direct object, or adjectival modifier etc.) [32]. Moreover, due to the lack of any phrasal-structure, dependency parse trees are amenable to free-word-order languages [32]. This allows us to model important word dependencies independent of their positional order (contrast this to context based text models for proximity and GREP-like search [2, 6, 26, 28] based on word-order). This flexibility allows us to design an indexing infrastructure that captures both text syntax and semantics in a concise manner.

KNITTIR Text Model. Let, $\mathcal{D} = \{d_1, d_2, \ldots, d_N\}$ denote a document collection, where, each document d contains a sequence of sentences: $d = \langle s_1, s_2, \ldots, s_{|d|} \rangle$. Also, each sentence, s, consists of a sequence of words: $s = \langle w_{[1,1]}, w_{[2,2]}, \ldots, w_{|s|,|s|} \rangle$, where, the words belong to the collection vocabulary, $\Sigma_{\mathcal{V}}$. We build a text model consisting of annotation layers $d_{\mathcal{L}}$ which impose additional semantics to the contiguous word sequence it annotates, $d_{\mathcal{L}} = \langle \ell_{[i,j]}, \ldots, \ell_{[p,q]} \rangle$ [26]. The word layer with its annotation alphabet, $\Sigma_{\mathcal{V}}$, is the basis layer, $d_{\mathcal{V}}$. The annotations are drawn from the annotator's alphabet, $\Sigma_{\mathcal{L}}$. Annotation alphabets for part-of-speech, coarse-grained named entities, temporal, and numerical expressions are denoted by $\Sigma_{\mathcal{P}}, \Sigma_{\mathcal{E}}, \Sigma_{\mathcal{T}}$, and $\Sigma_{\mathcal{N}}$, respectively. The syntactical text model [14] further allows

us to model non-contiguous relationships between words using the dependency parse tree, where the directed binary relationships ($\text{REL}_{\{i,j\}}$ or $\text{REL}_{\{H,D\}}$) between the head and dependent terms (w_i or w_H and w_j or w_D) belongs to $\Sigma_{\mathcal{G}}$. For example, in Figure 2 contains $\boxed{Williams} \xleftarrow{\text{NSUBJ}} \boxed{won}$, where the directed binary relation is NSUBJ ($\in \Sigma_{\mathcal{G}}$) with *won* as its head (w_H) and *Williams* as its dependent (w_D). Each annotation is formally modeled as [26]: $\ell \subset \mathbb{N} \times \mathbb{N} \times \Sigma_{\mathcal{L}}$, where $\mathbb{N} \times \mathbb{N}$ is represents the positional span on the basis layer while the $\Sigma_{\mathcal{L}}$ represents the annotation from its alphabet.

2.2 Analytics-Centric Search Framework

A central aspect for the user is the ease with which they can apply analytical operators much like native database operators. To this end, we reformulate the the traditional information retrieval (IR) search framework of *horizontal partitioned result set* wherein documents are returned in response to queries as a *vertical partitioned result set*. This analytics-centric search framework is described next.

An Analytics Workflow in KNITTIR takes as input a query consisting of a word sequence:

$$\text{Query:} \quad q = \langle w_1, w_2, \ldots, w_{|q|} \rangle. \tag{1}$$

Assuming that the query consists of a syntactically correct word sequence, we can obtain annotation for the query much like the documents in the collection. Thus, we can represent the query with the same layers of annotations as with KNITTIR's text model that includes its syntactical structure. KNITTIR matches queries to text regions in documents. A text region is defined as a contiguous sequence of words contained ($\sqsubset$) in a word layer of a document in the collection [26]:

$$\text{Text Region:} \quad w_{\langle i:j \rangle} = \langle w_i, w_{i+1}, \ldots, w_j \rangle \sqsubset d_{\mathcal{V}}. \tag{2}$$

A Vertically Partitioned Result Set refers to text regions that are structured and aligned vertically based on the syntactical, semantic, and contextual similarity with respect to the query. This ensures that the text region aligned to the query contains words that serve the same semantic function as in the query. Furthermore, the aligned words are syntactically and contextually similar as well with the words in the query. Formally, each tuple in the raw table represents a text region that is matched corresponding to the query operators applied to the queried word sequence:

$$\text{Raw Table:} \quad \bar{\mathcal{T}} = \bigcup \left(\text{DOC-ID}, A^1, A^2, \ldots, A^k \right). \tag{3}$$

	q	w_1	w_2	...	w_k	
	DOC-ID	A^1	A^2	...	A^k	A^{k+1}
r_1	id_1	a_1^1	a_1^2	...	a_1^k	a_1^{k+1}
r_2	id_2	a_2^1	a_2^2	...	a_2^k	a_2^{k+1}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
r_n	id_n	a_n^1	a_n^2	...	a_n^k	a_n^{k+1}

(4)

The vertically partitioned result set above thus represents the *two-dimensional text* as envisioned in [14]. In Equation 3, the number of attributes k for the structured representation of the text regions varies based on the query operator applied. For instance, if the similarity search operator is applied then the $k = |q|$, such that, we retrieve semantically similar text regions corresponding to the query. The rows of the raw table, $r \in \bar{\mathcal{T}}$, correspond to the number of matched text regions based on the operators applied on the query. We use the term *raw table* to reflect that the rows correspond to matches within the document collection. The tables generated by KNITTIR are in zero-normal form, as: $r \subset 2^{\Sigma_1} \times 2^{\Sigma_2} \times \ldots \times 2^{\Sigma_k}$, where, Σ refers to annotation alphabet for attribute A. This implies a cell in the raw table can potentially contain more than one annotation element drawn from its alphabet.

Reasoning over Vertical Partitions. The vertical partitioned result or the raw table $\bar{\mathcal{T}}$ can be further analyzed by allowing for operators that reason over the context (text regions) using syntactical and semantic annotations. Thus, we can augment the raw table $\bar{\mathcal{T}}$ by additional vertical partitions or attributes (i.e., A^{k+1} in Equation 4) by leveraging selective syntactical relations to specific semantic annotations.

2.3 Query Language

We now describe the key operators of KNITTIR's query language.

Similarity Search Operator ($\boxed{\sim}$) allows the user to retrieve similar text regions to the query based on syntactical, semantic, and contextual similarity (see Figure 1a). The syntactical similarity is processed by retrieving parts of the dependency parse tree corresponding to the query from the document collection. The semantic similarity is obtained by finding those words that occur with the same role for part-of-speech or coarse-grained named entity annotation for constituent parts of the query parse tree. Contextual similarity is arrived implicitly by combining syntactical and semantic similarity such that we identify sentences that contain similar words with same annotations and syntactical structure. Formally, the semantics can be expressed as:

$$\boxed{\sim} q = \bigcup \left\{ r \in \bar{\mathcal{T}} \;\middle|\; \begin{aligned} &\forall (i, j, \text{REL}_{\{i,j\}}) \in q \wedge \exists\, w_{\langle i:j \rangle} \sqsubset d_{\mathcal{V}} \\ &\qquad \wedge \text{REL}_{\{i,j\}} \sqsubset d_{\mathcal{G}} \\ &\wedge \ell_i \sqsubset (d_{\mathcal{P}} \vee d_{\mathcal{E}}) \wedge \ell_j \sqsubset (d_{\mathcal{P}} \vee d_{\mathcal{E}}) \\ &\qquad \wedge \left[w_i \oplus \ell_i \sqsubset d_{\mathcal{V}} \oplus (d_{\mathcal{P}} \vee d_{\mathcal{E}}) \vee \right. \\ &\qquad\quad \left. w_j \oplus \ell_j \sqsubset d_{\mathcal{V}} \oplus (d_{\mathcal{P}} \vee d_{\mathcal{E}}) \right] \end{aligned} \right\}, \tag{5}$$

where, the $\oplus$ implies the superimposition of the annotation layers such that the respective annotations are matched. In Equation 5, the $\boxed{\sim}$ operator produces a set of rows that correspond to text regions in the document collection, where the syntax of the query is matched. When performing the match of the syntax, a semantic match is also obtained such that a synonym substitution with same annotation is retrieved as only one of the arguments of the binary dependency relation needs to be contained.

Reasoning Operators provide the user the capability to group together entities (or noun phrases) into semantically related groups (e.g., *Williams* is an *athlete*), contrast them to other entities (e.g., *chess players* unlike *athletes* have more mental strain), or rank them using temporal and numerical expressions (e.g., *Williams* participated in the *2008 & 2012 Olympics*). Entities (or noun phrases) can be categorized, contrasted, or ranked using time and numbers based on *syntactical patterns* contained within the collection. We make this design choice so that KNITTIR can leverage the domain

knowledge contained in the document collection. An alternative design choice is to use external knowledge (dictionaries [55] or KGs [13]). However, this restricts reasoning operators in KNITTIR (e.g., limited to Wikipedia entities and categories). KNITTIR provides three types of reasoning operators: *categorize* (as GROUP △ and EXPAND ▽); *contrast* (as COMPARE ▷, and OUTLIER ↯); and *rank* (as TEMPORAL T and NUMERICAL N).

Categorize (GROUP △ & EXPAND ▽). KNITTIR provides two categorization operators: GROUP and EXPAND. The GROUP operator allows a user to determine categories for entities (or noun phrases) using collection-based syntactical patterns (see Figure 1b). Whereas, the EXPAND operator, allows a user to expand a category to a list of entities (or noun phrases) in the collection (see Figure 1c). Hearst Patterns [31] (see Figure 3a) capture such relationships between instances (hyponym or NP_I) and their categories (hypernym or NP_C). A pre-defined *syntactical pattern set* $\mathscr{P}_{\text{HEARST}}$ defines grouping and expansion rules for extraction of relevant noun phrases, entities, and categories from the document collection. The GROUP and EXPAND operators make use of the directed dependencies (i.e., NMOD:INCLUDING, NMOD:SUCH_AS, NMOD:LIKE, and NMOD:AS) or can reason by joining NMOD:OF, COP, and NSUBJ dependencies [11, 24, 31, 45, 46, 48, 49]. Table 1 shows the pattern set $\mathscr{P}_{\text{HEARST}}$. Formally, the GROUP and EXPAND operators take as input relevant annotated cells corresponding to part-of-speech or entity attributes (i.e., $\mathtt{a} = w \oplus \ell$, where $\ell \in \{\mathrm{d}_\mathcal{P} \vee \mathrm{d}_\mathcal{E}\}$) from the raw table $\bar{\mathcal{T}}$ and extract grouping categories or expanded entity sets as augmentation for that row:

$$\text{△}(w \oplus \ell) = \left\{ w_{\langle \mathrm{i:j} \rangle} \sqsubset \mathrm{d}_\mathcal{V} \;\middle|\; \begin{array}{r} \exists \mathrm{p} \in \mathscr{P}_{\text{HEARST}} \wedge \forall (\mathrm{i},\mathrm{j}, \text{REL}_{\{\mathrm{i},\mathrm{j}\}}) \in \mathrm{p} \\ \wedge\, \text{REL}_{\{\mathrm{i},\mathrm{j}\}} \sqsubset \mathrm{d}_\mathcal{G} \\ \wedge\, \ell_\mathrm{i} \sqsubset (\mathrm{d}_\mathcal{P} \vee \mathrm{d}_\mathcal{E}) \wedge\, \ell_\mathrm{j} \sqsubset (\mathrm{d}_\mathcal{P} \vee \mathrm{d}_\mathcal{E}) \\ \wedge \left[w_\mathrm{i} \oplus \ell_\mathrm{i} \sqsubset \mathrm{d}_\mathcal{V} \oplus (\mathrm{d}_\mathcal{P} \vee \mathrm{d}_\mathcal{E}) \vee \right. \\ \left. w_\mathrm{j} \oplus \ell_\mathrm{j} \sqsubset \mathrm{d}_\mathcal{V} \oplus (\mathrm{d}_\mathcal{P} \vee \mathrm{d}_\mathcal{E}) \right] \\ \wedge\, \exists \bar{w} \oplus \bar{\ell} = w \oplus \ell \end{array} \right\}. \quad (6)$$

In the above equation, the GROUP operator expands an entity present in a raw table cell using patterns in $\mathscr{P}_{\text{HEARST}}$ to match text regions containing categories to the queried entity (or noun phrase).

Contrast (COMPARE ▷ & OUTLIER ↯). The COMPARE operator allows users to compare entities conveyed in text using comparative or superlative parts-of-speech. Whereas, the OUTLIER operator highlights implicit commonsense knowledge by extracting explicit exceptions (outliers). We describe the contrastive operators next.

Compare (▷). Relationships between entities is often contrasted using a finite set of adjectives [16]. Reasoning over such contrasting relationships helps us provide comparative analytics over entities. This is particularly useful for sport journalists or political scientists when drawing comparisons between persons (see Figure 3b). To support such analytics, the COMPARE ▷ operator augments an entity or noun phrase attribute with other entities using their comparative relations. Specifically, the COMPARE operator uses comparative and superlative adjectives in addition to the join of NSUBJ, COP, and OBL:THAN dependency relations (see Figure 3b). Table 2 shows the comparative relational patterns $\mathscr{P}_{\text{COMPARE}}$ from [16]. The semantics for COMPARE can be specified in a manner similar to Equation 6 with the $\mathscr{P}_{\text{COMPARE}}$ pattern set.

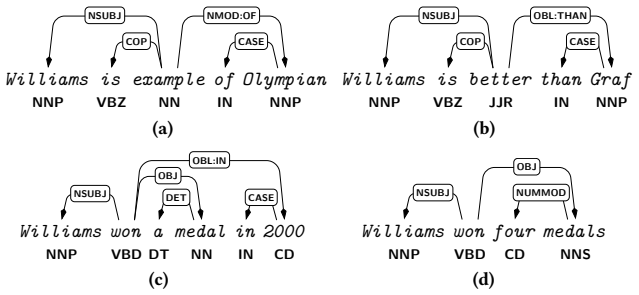

Figure 3: Annotated text regions containing patterns for (a) CATEGORIZATION (b) COMPARE (c) TEMPORAL and (d) NUMERICAL operators.

Table 1: Pattern set $\mathscr{P}_{\text{HEARST}}$ for relationships between instances (hyponym or $w_\mathrm{I} \oplus \text{NP}_\mathrm{I}$) and their categories (hypernym or $w_\mathrm{C} \oplus \text{NP}_\mathrm{C}$).

HEARST PATTERN	DEPENDENCY PATTERN
$w_\mathrm{C} \oplus \text{NP}_\mathrm{C}$ *including* $w_\mathrm{I} \oplus \text{NP}_\mathrm{I}$	$\langle w_\mathrm{C} \oplus \text{NP}_\mathrm{C}$, NMOD:INCLUDING, $w_\mathrm{I} \oplus \text{NP}_\mathrm{I} \rangle$
$w_\mathrm{C} \oplus \text{NP}_\mathrm{C}$ *such as* $w_\mathrm{I} \oplus \text{NP}_\mathrm{I}$	$\langle w_\mathrm{C} \oplus \text{NP}_\mathrm{C}$, NMOD:SUCH_AS, $w_\mathrm{I} \oplus \text{NP}_\mathrm{I} \rangle$
$w_\mathrm{C} \oplus \text{NP}_\mathrm{C}$ *like* $w_\mathrm{I} \oplus \text{NP}_\mathrm{I}$	$\langle w_\mathrm{C} \oplus \text{NP}_\mathrm{C}$, NMOD:LIKE, $w_\mathrm{I} \oplus \text{NP}_\mathrm{I} \rangle$
$w_\mathrm{I} \oplus \text{NP}_\mathrm{I}$ *as* $w_\mathrm{C} \oplus \text{NP}_\mathrm{C}$	$\langle w_\mathrm{C} \oplus \text{NP}_\mathrm{C}$, NMOD:AS, $w_\mathrm{I} \oplus \text{NP}_\mathrm{I} \rangle$
$w_\mathrm{I} \oplus \text{NP}_\mathrm{I}$ [*is*‖*was*‖*are*‖*were*] $w_\mathrm{C} \oplus \text{NP}_\mathrm{C}$	$\langle w_\mathrm{C} \oplus \text{NP}_\mathrm{C}$, NSUBJ, $w_\mathrm{I} \oplus \text{NP}_\mathrm{I} \rangle$ $\langle w_\mathrm{C} \oplus \text{NP}_\mathrm{C}$, COP, *is*‖*was*‖*are*‖*were*$\rangle$
$w_\mathrm{I} \oplus \text{NP}_\mathrm{I}$ [*are*‖*is*] *examples of* $w_\mathrm{C} \oplus \text{NP}_\mathrm{C}$	$\langle$*example*, NMOD:OF, $w_\mathrm{C} \oplus \text{NP}_\mathrm{C} \rangle$ $\langle$*example*, COP, *is*‖*are*$\rangle$ $\langle$*example*, NSUBJ, $w_\mathrm{I} \oplus \text{NP}_\mathrm{I} \rangle$

Table 2: Pattern set $\mathscr{P}_{\text{COMPARE}}$. Comparative or superlative part-of-speech annotations such as JJR, JJS, RBR, and RBS are marked with •.

COMPARATIVE PATTERN	DEPENDENCY PATTERN
NN_H • *than* NN_D	$\langle w_\text{A} \oplus \bullet$, NSUBJ, $w_\text{H} \oplus \text{NN}_\text{H} \rangle$ $\langle w_\text{A} \oplus \bullet$, COP, *is*‖*was*‖*are*‖*were*$\rangle$ $\langle w_\text{A} \oplus \bullet$, OBL:THAN, $w_\text{D} \oplus \text{NN}_\text{D} \rangle$
NN_H • *than* NN_D	$\langle w_\text{A}$, NSUBJ, $\text{NN}_\text{H} \rangle$ $\langle w_\text{A}$, COP, *is*‖*was*‖*are*‖*were*$\rangle$ $\langle w_\text{A}$, ADVMOD, $w_\text{B} \oplus \bullet \rangle$ $\langle w_\text{B}$, OBL:THAN, $\text{NN}_\text{D} \rangle$

Table 3: Pattern set $\mathscr{P}_{\text{TIME}}$. NER refers to PERSON, ORG., or LOC.

TEMPORAL PATTERN	DEPENDENCY PATTERN
$w_\text{H} \oplus$ NER NOUN-PHRASE PREP. $w_\text{D} \oplus$ DATE	$\langle w_\text{A} \oplus \text{NN}_\text{A}$, NSUBJ , $w_\text{D} \oplus \text{NER}_\text{H} \rangle$ $\langle w_\text{A} \oplus \text{NN}_\text{A}$, NMOD:PREP., $w_\text{D} \oplus \text{DATE}_\text{D} \rangle$
$w_\text{H} \oplus$ NER VERB-PHRASE PREP. $w_\text{D} \oplus$ DATE	$\langle w_\text{A} \oplus \text{NN}_\text{A}$, NSUBJ , $w_\text{D} \oplus \text{NER}_\text{H} \rangle$ $\langle w_\text{A} \oplus \text{NN}_\text{A}$, OBL:PREP. , $w_\text{D} \oplus \text{DATE}_\text{D} \rangle$

Outliers (↯). Commonsense reasoning from large document collections is challenging as implicit knowledge concerning entities and events is never made explicit in text. However, entities or events that defy such commonsense or natural knowledge is usually surprising and made explicit in text [42]. For instance, in the sentence, "*although, a doctor, Debra Thomas won a medal in 1988 Olympics*", we can identify the implicit knowledge that doctors do not participate in Olympics. We provide the OUTLIER operator, to assist users in extracting text regions that contain such peculiarities thereby highlighting the commonsense knowledge (see Figure 1d). The OUTLIER operator allows a user to expand the category with phrases extracted from the document collection that do *not* usually belong to the category [42]. The pattern set of defeasible statements [42, 51], $\mathscr{P}_{\text{OUTLIER}}$, consists of direct binary relationships of conjunction or adverbial clause modifiers: CONJ:BUT, ADVCL:ALTHOUGH, ADVCL:ALBEIT, ADVCL:DESPITE, and ADVCL:IF. Using $\mathscr{P}_{\text{OUTLIER}}$, we can extract corresponding phrases and entities that deviate from the expected. The semantics of the OUTLIER can be specified in a manner similar to Equation 6 with $\mathscr{P}_{\text{OUTLIER}}$.

Ranking (TIME [T] & NUMBER [N]). Temporal and numerical values are annotated as a CARDINAL VALUE (CD) by the part-of-speech annotator. These cardinal values can be further resolved to their coarse-grained named entity types (e.g., TIME, DATE, or MONEY) and furthermore to precise intervals via additional annotators (e.g., temporal taggers such as SUTime [18]). The operators [T] and [N] allow a user to reason about temporal and numerical expressions in relation to the entities (or noun phrases) in the raw table (see Figure 1f and 1g). This allows a user to rank (or sort) the augmented table for identifying answers to several temporal and numerical relations such as: after, before, soon, first, second, and last.

Temporal Reasoning Operator [T]. Temporal expressions are annotated as DATE, TIME, or DURATION entities and further resolved to time intervals via temporal annotators. Moreover, temporal expressions are connected to either a verbal phrase or a noun phrase through prepositions: *from*, *in*, *during*, *since*, *until*, and *between*. Expressions connected to a verbal phrase are annotated with a OBL dependency relationships (see Figure 3c). While, expressions connected to noun phrases are annotated with a NMOD dependency relationship [47]. To extract time intervals from the collection we create a pattern set of such relations $\mathscr{P}_{\texttt{TIME}}$ (see Table 3). The semantics of the temporal reasoning operator [T] are:

$$\boxed{\texttt{T}}(w \oplus \ell) = \left\{ [\mathrm{b}, \mathrm{e}] \sqsubset \mathrm{d}_{\mathcal{T}} \;\middle|\; \begin{array}{r} \exists \mathrm{p} \in \mathscr{P}_{\texttt{TIME}} \wedge \forall (\mathrm{i}, \mathrm{j}, \texttt{REL}_{\{\mathrm{i},\mathrm{j}\}}) \in \mathrm{p} \\ \wedge \texttt{REL}_{\{\mathrm{i},\mathrm{j}\}} \sqsubset \mathrm{d}_{\mathcal{G}} \\ \wedge\, \ell_{\mathrm{i}} \sqsubset (\mathrm{d}_{\mathcal{P}} \vee \mathrm{d}_{\mathcal{E}}) \wedge\, \ell_{\mathrm{j}} \sqsubset (\mathrm{d}_{\mathcal{P}} \vee \mathrm{d}_{\mathcal{E}}) \\ \wedge \left[w_{\mathrm{i}} \oplus \ell_{\mathrm{i}} \sqsubset \mathrm{d}_{\mathcal{V}} \oplus (\mathrm{d}_{\mathcal{P}} \vee \mathrm{d}_{\mathcal{E}} \vee \mathrm{d}_{\mathcal{T}}) \vee \right. \\ \left. w_{\mathrm{j}} \oplus \ell_{\mathrm{j}} \sqsubset \mathrm{d}_{\mathcal{V}} \oplus (\mathrm{d}_{\mathcal{P}} \vee \mathrm{d}_{\mathcal{E}} \vee \mathrm{d}_{\mathcal{T}}) \right] \\ \wedge\, \exists \bar{w} \oplus \bar{\ell} = w \oplus \ell \end{array} \right\}. \quad (7)$$

In the equation above, for an entity (or noun phrase) in the raw table, the temporal reasoning operator [T] extracts time intervals in the collection using the patterns in $\mathscr{P}_{\texttt{TIME}}$.

Numerical Reasoning Operator [N]. Numerical expressions are classified by the coarse-grained named entity recognizer with MONEY, PERCENT, NUMBER, or ORDINAL annotation. Like temporal expressions, numerical expressions can also be resolved to intervals using additional annotators. However, unlike temporal expressions, numerical values are often accompanied by units (e.g., *two medals*), which is annotated as a NUMMOD relationship by the dependency parser (see Figure 3d). Moreover, also unlike temporal expressions, to extract the numerical values and units related to an entity (or noun phrase) requires reasoning over multiple triples in the dependency parse. Concretely, the numerical reasoning operator applied to an entity (or noun phrase) $\mathtt{a} = w \oplus \ell$ (i.e., $\boxed{\texttt{N}}(w \oplus \ell)$) maps to a set of numerical intervals with units, $\bigcup ([\mathrm{b}, \mathrm{e}]; \texttt{UNIT})$, by reasoning over the dependency parse tree of a sentence, where $\mathtt{a} = w \oplus \ell$ is its nominal subject (NSUBJ) and has path to a numerical expression (MONEY, PERCENT, NUMBER, or ORDINAL) with its unit (related via NUMMOD).

2.4 Index Design

We now describe five index design choices and discuss which one can support similarity search and reasoning operators such that they can be processed quickly.

Inverted Indexes over words work well for keyword-based search. To use this design choice to support similarity search and reasoning operators, inverted indexes over elements from the different annotation layers can be built [7, 26, 56]. However, with only inverted indexes over individual annotation elements, we may retrieve many false positives, e.g., for syntactical relationships between words, we need to additionally verify the matches using a direct index that stores the complete annotated document [40]. The shortcoming of combinations of annotated inverted indexes has been evaluated before [26].

Graph Indexes seem a natural design choice for retrieval of the parse tree corresponding to the query. Graph indexing techniques have been employed successfully for search of semi-structured data such as XML and JSON [33]. A graph index for the parse tree would entail enumerating paths from the root of the dependency path tree to the leaf. To support efficient query processing, paths of fixed length can be enumerated and indexed [57]. Indeed, graph indexes have been used for searching collections annotated for dependency parse trees [55]. The authors index parse trees using an index design based on data guides. Specifically, their storage scheme is implemented using a relational database. However, this approach is ill-suited for KNITTIR, as sentences that are similar may not necessarily have same sequence of relations from the root node of the dependency parse tree. This shortcoming is reflected in [55] as the authors rely on external knowledge (i.e., word embeddings and pre-computed dictionaries) to establish paraphrases for semantic similarity. This is prohibitive in domain adaptability of the storage scheme. Furthermore, for reasoning operators, we need the flexibility of locating matches that combine the dependency parse tree relationships with other annotation layers of the document. Thus, path enumeration based design choice is too limited for KNITTIR.

2-Stitch and 2-Fragment Indexes. Naïve element-wise inverted indexes for storing dependency parse trees are expensive when performing query processing as further verification using direct indexes is needed. On the other hand, graph indexing techniques based on path enumeration are too brittle to support similarity search. Between two extremes, we consider a combination of two other design choices. We first consider 2-stitch and 2-fragment indexes that store aligned and staggered combinations of words from the basis layer and an additional annotation layer [26] (where 2 indicates the number of layers considered). 2-stitch and 2-fragment indexes significantly reduce times for performing GREP-like search using word sequences and annotations. For instance, in the annotated sentence in Figure 2, an example of 2-fragment is the aligned combination of the word *two* and the coarse-grained named entity NUMBER. An example of 2-stitch is the staggered combination of the word *during* and the coarse-grained named entity DATE. However, 2-stitch and 2-fragment indexes are still insufficient for storage of syntactical patterns that are necessary for similarity search. This is because, just like with the inverted index design, we need to resort to the direct index for ensuring the correct syntactical match for the query (i.e., dependency relations between words and annotations).

Weave Indexes. All three design alternatives fail to provide us a comprehensive storage scheme to support KNITTIR's query operators as we have to resort to the direct index to verify either the syntax, semantics, or the context of the matched text region. In order to retrieve matching text regions quickly, we require a feasible solution

that accommodates the syntax, semantics, and the context of a word in the annotated text model. A key limiting factor in previous design choices is the inability to store the syntactical information conveyed by the dependency parse. The graph-based indexing approaches (e.g., [55]) consider a rigid path enumeration based approach. While 2-stitch and 2-fragment based approach [26] ignores the syntactical aspect completely by enumerating all possible combinations. As highlighted in Section 2.1, dependency parse trees allow us to model the syntax of a sentence as a collection of binary relationships. The dependency parse in contrast to constituency parse does not rely on a context-free grammar resulting in phrasal structure. Thus, we can reduce the path enumeration to simple sets of binary relationships between the words [14, 32]. Furthermore, we can combine this with the 2-fragment based index design. That is, deriving binary relationships in combination with a variable superimposition of the finest or coarsest-granular semantics available for a word to arrive at patterns that combine both the syntax and semantics in the annotated text model (e.g., variable semantics superimposition of PERSON or NNP on *Williams* in Figure 2). Thus, KNITTIR considers a design choice of deriving sets of binary relationships that combine both syntax and semantics from the dependency parse tree using the *holing process* [14]. Based on this holing process, we create weave indexes that record combinations of words and other aligned annotations related by binary relationship obtained from the dependency parse tree. The weave indexes also help us consider only syntactically related word and annotation combinations whereas 2-stitch indexes consider *all* n-gram and annotation combinations; many of which many not be syntactically relevant. Based on the weave index, we can retrieve words that are relevant for syntactical and semantic similarity based on the part-of-speech role, named entity, or other annotation in addition to their correct syntactical relationship. For example, to index the dependency relation pattern NMOD:IN in Figure 2, we consider the 2-fragments *Olympics*⊕MISC. and *Sydney*⊕LOC.) by superimposing the basis layer and coarse-grained named entity layer. After applying the holing process, we get two weave indexing patterns: $\boxed{\textit{Olympics}\oplus\text{MISC.}} \xrightarrow{\text{NMOD:IN}} \boxed{\text{LOC.}}$ and $\boxed{\text{MISC.}} \xrightarrow{\text{NMOD:IN}} \boxed{\textit{Sydney}\oplus\text{LOC.}}$. For each indexing pattern, we record the holed word (e.g., *Sydney* for the first weave pattern), its sentence identifier, and its positional span in the document. The holed word is dictionary encoded and the posting list payloads are compressed using Patched Frame of Reference [5, 59].

Aggregate Weave Indexes. Weave indexes help us support similarity search operator $\boxed{\sim}$ in a scalable manner. Weave indexes can also support categorical (GROUP $\boxed{\triangle}$ and EXPAND $\boxed{\triangledown}$), contrastive (COMPARE $\boxed{\triangleright}$ and OUTLIER $\boxed{\not\triangleright}$), and rank (TIME $\boxed{\text{T}}$ and NUMBER $\boxed{\text{N}}$) reasoning operators. However, using only weave indexes can slow down the processing of reasoning operators as they require lookup of multiple syntactical patterns in their pattern sets $\mathscr{P}_\bullet$. To speed up the processing of reasoning KNITTIR queries, we create *aggregate weave indexes* that record unified reasoning results over multiple syntactical patterns. For example, in Table 2 for the COMPARE operator, we can aggregate the result of the multiple dependency parse triples into a unified pattern: $\langle \text{NN}_\text{H}, \text{COMPARE}, \text{NN}_\text{D} \rangle$. Similar to the weave indexes, we apply the holing process to index this aggregated pattern with the holed word, its sentence identifier, and its positional span as part of the payload of the postings list.

Input: query sentence (s).
Output: matched text regions within sentences (L).

```
1:  procedure SimilaritySearch(s)
2:    R ⟵ parse(s)                                  ▷ dependency parse of s
3:    map ⟵ ∅                                       ▷ initialize key-value map
4:    for w ∈ s do                                                ▷ word in S
5:      for (w_H ⊕ ℓ_H, REL, w_D ⊕ ℓ_D) ∈ R do ▷ dependency in relations
6:        if w_H = w then                    ▷ dependency head equals word
7:          searchKey ⟵ (ℓ_H, REL, w_D ⊕ ℓ_D)                ▷ omit head
8:          L ⟵ retrieve values for searchKey
9:          map.put(w_H ⊕ ℓ_H, L)
10:       else if w_D = w then ▷ dependency dependent equals word
11:         searchKey ⟵ (w_H ⊕ ℓ_H, REL, ℓ_D)          ▷ omit dependent
12:         L ⟵ retrieve values for searchKey
13:         map.put(w_D ⊕ ℓ_D, L)
14:   L ⟵retrieve list of value for map[0]
15:   L′ ⟵ ∅                                          ▷ temporary variable
16:   for (i ⟵ 1;i < map.size();i + +) do
17:     L′ ⟵ retrieve list of value for map[i]
18:     L ⟵ Intersect L & L' for matches within a sentence
19:   return L
```

Algorithm 1: Similarity Search Operator

2.5 Query Processing

An Analytics Workflow in KNITTIR starts with a user input query consisting of a word sequence, $q = \langle w_1, w_2, \ldots, w_{|q|} \rangle$. This initial query is first processed for the same annotation layers as the documents (see Figure 2). With the annotated query, we can retrieve similar text regions, which are then vertically partitioned into a raw table. A user can then further perform reasoning operations (e.g., GROUP $\boxed{\triangle}$) on the structured table by retrieving further attributes from the annotated document collection for analytics (e.g., entity categories). We next explain how the query processing for the semantic similarity and reasoning operators is done efficiently with the help of weave and aggregated weave indexes.

The Similarity Search operator ($\boxed{\sim}$) is processed by KNITTIR using combinations of semantic, syntactic, and contextual similarity. To retrieve semantically similar text regions, KNITTIR superimposes the most fine granular annotation corresponding to the word. For instance, in Figure 2, the query term *Williams* can refer to either a proper noun (NNP) as a part-of-speech or a PERSON as coarse-grained named entity. To process the term *Williams*, we choose here the most fine-granular annotation PERSON resulting in the 2-fragment *Williams*⊕PERSON. The syntactical similarity is obtained by combining the 2-fragments with binary dependency parse relationships (e.g., in Figure 2 we have, $\boxed{\textit{Williams}\oplus\text{PERSON}} \xleftarrow{\text{NSUBJ}} \boxed{\textit{won}\oplus\text{VBD}}$). The contextual similarity is obtained implicitly by combining semantic and syntactical similarity. Contextual similarity identifies words in sentences that provide the same semantics and syntax as those referred to by the 2-fragments and *also* the dependency parse relationships (e.g., *claimed* in $\boxed{\textit{Williams}\oplus\text{PERSON}} \xleftarrow{\text{NSUBJ}} \boxed{\textit{claimed}\oplus\text{VBD}}$ provides the same semantics and syntax as *won* in Figure 2). To process the similarity search operator only the weave indexes are required. The query is processed (see Algorithm 1) by enumerating all the weave patterns from the input query. The is done by selecting all the binary dependency parse relations and alternatively omitting the words from the basis layer. Thus, each dependency parse relation can enumerate two weave patterns (e.g., for the relation NSUBJ in Figure 2, two weave patterns are generated $\boxed{\text{PERSON}} \xleftarrow{\text{NSUBJ}} \boxed{\textit{won}\oplus\text{VBD}}$

and $\boxed{Williams \oplus \text{PERSON}} \xleftarrow{\text{NSUBJ}} \boxed{\text{VBD}}$). The intersection of the weave patterns that lie within sentence boundaries provides us matches for the similarity search operator. The text regions retrieved as part of the similarity search operator are vertically partitioned into tables using the substitutions obtained as part of the weave patterns. To vertically partition the text regions, a table is instantiated equal to the query length. The cell values for each row then corresponds to the words that were retrieved as part of the weave patterns that locate the similarity match for the query.

Reasoning Operators retrieve additional attributes for entities (or noun phrases) in the raw table generated by the similarity search operator. Reasoning operators, i.e., (GROUP $\boxed{\triangle}$ and EXPAND $\boxed{\triangledown}$), contrastive (COMPARE $\boxed{\triangleright}$ and OUTLIER $\boxed{\not\triangleright}$), and rank (TIME $\boxed{\text{T}}$ and NUMBER $\boxed{\text{N}}$) correspond to those that we have defined as part of our query language (see Section 2.3). These operators are readily resolvable by looking up the expansions from the aggregated weave indexes.

3 Evaluation

We now describe the setup and results of our experiments.

Annotated Document Collections. We annotated and indexed three large document collections with KNITTIR. The first and the smallest collection, consists of twenty years' (1987-2007) worth of news articles available as the New York Times Annotated Corpora [9]. The second collection is the English Wikipedia [3] which consists of encyclopedic articles concerning real-world entities and events. The third and final collection is the news-centric subset of the cleaned English Common Crawl [1] available as C4-News [43]. We processed all collections with the Stanford CoreNLP suite of semantic annotators [37]. In particular, we annotated the documents for part-of-speech; coarse-grained named entities; resolved temporal and numerical expressions; and enhanced dependency parse trees. Collection statistics are shown in Table 4.

Implementation and Indexes. The implementation of KNITTIR was carried out from scratch in Java. The annotation of the document collections and their indexing is carried out in a distributed manner over our Hadoop cluster of twenty five machines. Each machine is equipped with at least an Intel Xeon CPU with 16 cores clocked at 2.20 GHz and at least of 128 GB of RAM. We index dependency parse triples for sentence length up to 100 words. This is because sentences longer than this limit are not grammatically correct and correspond to lists, tables, code excerpts or other metadata elements that may have been omitted during boilerplate removal by the collection provider [10]. The indexes are stored in HBase, a distributed key-values store, running on our cluster. Index sizes and their build times are shown in Table 5 and 6, respectively.

Query Testbed. To evaluate the efficiency of KNITTIR for analytical tasks, we construct a testbed of sentences describing important events from "The New York Times - On This Day" portal [8]. The testbed sentences are then used to start the analytics workflow by identifying similar sentences in the document collection and subsequently to apply the reasoning operators. An example sentence is: "*The Summer Olympics opened in Sydney, Australia.*" (quoted from [8]). In total, we have 4,875 sentences in our testbed.

Baselines. As a naïve baseline (SCAN), we first observe the time taken to scan the document collections in a distributed manner over our Hadoop cluster. This gives a good estimate of how the indexes of the system or the baseline compares to a simple distributed scan. The central operator for retrieval of contextually, syntactically, and semantically similar sentences is the similarity search $\boxed{\sim}$ operator. We compare the the processing of $\boxed{\sim}$ operator of KNITTIR with three baselines: WAND, STICH, and FAST. These baselines are aligned with the design choices discussed in Section 2.4. Concretely, the first baseline, WAND, processes the $\boxed{\sim}$ operator contextually by locating text regions using inverted indexes over the word and annotation layers. To further verify that the retrieved text regions match the syntactical and semantic similarity, we create a direct index where the complete representation of the document with all of its annotation layers are stored. Thus, for locating portion of a weave, $\boxed{Williams \oplus \text{PERSON}} \xleftarrow{\text{NSUBJ}} \boxed{won \oplus \text{VBD}}$, WAND computes: $\boxed{\pi_1}$ (PERSON $\boxed{.*?}$ *won*), where $\boxed{.*?}$ matches all text regions lazily (i.e., shortest possible text region match) that contain combinations of a PERSON in the coarse named entity layer and *won* in the word layer (while respecting the weave nodes positional order) and $\boxed{\pi_1}$ subsequently projects them to lie within a sentence. The second baseline, STICH, corresponds to instantiating 2-stitch indexes that record staggered combinations of words and annotations thus speeding the spotting of text regions. Similar to WAND, STICH can process the $\boxed{\sim}$ operator contextually, while it relies on the direct index for verifying the syntactical and semantic similarity. The third baseline, FAST, corresponds to instantiating the 2-fragment indexes in addition to the 2-stitch index to help resolve semantic similarity. Thus, for locating portion of a weave, $\boxed{Williams \oplus \text{PERSON}} \xleftarrow{\text{NSUBJ}} \boxed{won \oplus \text{VBD}}$, FAST computes $(Williams \oplus \text{PERSON} \leftarrow won)$ using the combination of 2-stitch and 2-fragment indexes. The syntactical similarity, just like WAND and STICH, is verified using the direct index. We further evaluate the efficiency of creating AGGREGATED WEAVE indexes for processing reasoning operators when compared to processing the same reasoning operator using the WEAVE indexes. A summary of which indexes the baselines and our systems use is given in Table 7.

Table 4: Raw collection sizes and their annotation statistics. The statistic for dependency parse corresponds to the number of triples.

	SIZE (GB)	#DOCS	#WORD	#NE	#TIME	#NUM	#PARSE
NYT	3.06	1.86 M	1.06 B	107.77 M	15.41 M	21.72 M	1.03 B
EN-WIKI	34.44	6.34 M	3.81 B	626.27 M	150.89 M	115.14 M	3.62 B
C4-NEWS	14.38	13.81 M	6.14 B	572.60 M	85.18 M	113.77 M	6.02 B

Table 5: Index sizes for the document collections.

INDEX TYPE	NYT	EN-WIKI	C4-NEWS
WORD	6.7 GB	21.1 GB	37.7 GB
ANNOTATION	1.7 GB	5.3 GB	10.7 GB
FRAGMENT	9.5 GB	25.8 GB	40.6 GB
STITCH	179.8 GB	374.9 GB	2.2 TB
WEAVE	114.7 GB	329.1 GB	805.8 GB
AGGREGATED WEAVE	142.4 GB	582.1 GB	1.0 TB
DIRECT	25.1 GB	96.4 GB	143.0 GB

Table 6: Index build times (hours:minutes:seconds:milliseconds).

INDEX TYPE	NYT	EN-WIKI	C4-NEWS
WORD	00:06:07:614	00:44:29:830	00:32:47:873
ANNOTATION	00:05:40:049	01:00:21:072	00:27:41:374
FRAGMENT	00:06:55:290	01:03:02:463	00:37:11:780
STITCH	03:10:13:657	04:43:16:730	09:24:44:724
WEAVE	00:16:12:800	11:47:28:081	2:28:41:297
AGGREGATED WEAVE	00:22:59:482	13:30:17:999	5:26:44:965
DIRECT	00:22:47:283	00:36:32:570	00:32:25:654

Table 7: Indexes used by the baselines.

SYSTEM	N-GRAM INDEXES	ANNOTATION INDEXES	STITCH INDEX	FRAGMENT INDEX	DIRECT INDEX
WAND	•	•	×	×	•
STICH	×	×	•	×	•
FAST	×	×	•	•	•

Table 8: Runtimes in seconds for a distributed scan.

	NYT	EN-WIKI	C4-NEWS
SCAN	74	321	283

Table 9: Query processing runtimes (in seconds) for cold caches to process the similarity search operator ~.

SYSTEM	NYT	EN-WIKI	C4-NEWS
WAND	294.85 ± 112.81	1420.83 ± 602.59	3008.44 ± 1347.91
STICH	30.94 ± 30.85	136.22 ± 279.12	276.85 ± 208.91
FAST	48.73 ± 41.70	195.75 ± 246.81	351.45 ± 352.73
WEAVE	6.50 ± 7.30	24.55 ± 32.69	75.14 ± 86.11

Table 10: End-to-end runtimes (in seconds) for cold caches to process the similarity search operator ~.

SYSTEM	NYT	EN-WIKI	C4-NEWS
WAND	303.50 ± 111.09	1494.22 ± 732.40	2898.66 ± 1272.97
STICH	34.52 ± 35.17	212.88 ± 665.32	302.80 ± 248.81
FAST	51.26 ± 45.91	261.76 ± 558.25	380.41 ± 380.07
WEAVE	6.21 ± 7.25	25.07 ± 32.85	75.78 ± 86.58

Setup. We evaluate the systems for the time it takes to process each of the operators in an analytics workflow. To do so, we sample 100 queries to initialize the analytics workflow. First, we measure the runtime for the systems to process the similarity search operator ~. Second, we measure the time to execute the reasoning operators using the WEAVE indexes and the AGGREGATED WEAVE indexes for a sample of 100 unique entities in the initial sample of 100 queries. We evaluate each query three times to measure the runtimes in cold-caches, which are simulated by shuffling the query sample in between rounds. As analytics-centric tasks are complete recall-oriented, we retrieve complete results sets for all the baselines and systems (unlike nearest-neighbor search in vector-based indexes).

Results for Distributed Scan are shown in Table 8. As can be seen, the results for the collections range in the order of few minutes, with Wikipedia taking the longest. This can be attributed to the observation the Wikipedia documents on average are longer as they contain more words as compared to the other two collections.

Results for Similarity Search operator ~ are shown in Tables 9 and 10. To process the similarity search operator ~, all the systems annotate the query, process the query, and generate the raw table. The first set of results (see Table 9) show the time taken to annotate and process a query *without* generating the raw table, which requires additional consultations from the direct index for the three baselines. While, the second set of results (see Table 10), show the complete end-to-end runtime results. Overall, comparing the results in Table 9 and 10, we observe that for the three baselines (i.e., WAND, STICH, and FAST) we incur additional time for materializing the raw table by looking up the synonyms for the results in the direct index. Concretely, we see that for WAND on average 41.02 seconds need to be spent on materializing the table for NYT and EN-WIKI. For, C4-NEWS in the case of WAND, due to the high standard deviation in query processing in a distributed setting takes more time than the end-to-end runtimes. While, for STITCH and FAST, we see an average of additional 35.40 and 32.50 seconds being spent on

Table 11: End-to-end runtimes (in seconds) for reasoning tasks.

RUNTIME RESULTS FOR GROUP OPERATOR △.			
SYSTEM	NYT	EN-WIKI	C4-NEWS
WEAVE	12.57 ± 9.93	51.18 ± 27.59	98.11 ± 45.36
AGGREGATE-WEAVE	0.09 ± 0.19	0.84 ± 1.01	0.51 ± 0.98
RUNTIME RESULTS FOR EXPAND OPERATOR ▽.			
SYSTEM	NYT	EN-WIKI	C4-NEWS
WEAVE	2.04 ± 5.30	4.60 ± 9.30	8.28 ± 13.97
AGGREGATE-WEAVE	0.06 ± 0.14	0.61 ± 0.72	0.38 ± 0.56
RUNTIME RESULTS FOR OUTLIER OPERATOR ↯.			
SYSTEM	NYT	EN-WIKI	C4-NEWS
WEAVE	0.01 ± 0.03	0.01 ± 0.00	0.03 ± 0.02
AGGREGATE-WEAVE	0.01 ± 0.04	0.02 ± 0.01	0.02 ± 0.03
RUNTIME RESULTS FOR COMPARE OPERATOR ▷.			
SYSTEM	NYT	EN-WIKI	C4-NEWS
WEAVE	416.30 ± 740.83	1513.89 ± 2219.84	–
AGGREGATE-WEAVE	0.01 ± 0.02	0.04 ± 0.04	0.04 ± 0.04
RUNTIME RESULTS FOR TEMPORAL OPERATOR T.			
SYSTEM	NYT	EN-WIKI	C4-NEWS
WEAVE	69.41 ± 116.18	531.24 ± 989.01	961.43 ± 1192.36
AGGREGATE-WEAVE	0.01 ± 0.02	0.07 ± 0.07	0.05 ± 0.07
RUNTIME RESULTS FOR NUMERICAL OPERATOR N.			
SYSTEM	NYT	EN-WIKI	C4-NEWS
WEAVE	672.52 ± 752.27	2768.87 ± 3365.56	–
AGGREGATE-WEAVE	0.01 ± 0.03	0.04 ± 0.05	0.05 ± 0.11

materializing the table over the three collections. For our system that utilizes WEAVE indexes, there is no sizeable difference between the query processing and the end-to-end runtimes, as KNITTIR can materialize the raw table by looking up the synonyms stored in the posting lists itself without consulting the direct index.

We now discuss the speedup obtained by utilizing WEAVE indexes over the three baselines for processing the similarity search operator ~. First, we consider the speedup over WAND, which performs similarity search using indexes over words and annotations. We observe that WEAVE indexes provide speedups in the range of 38-60× compared to WAND over the three collections. This significant speedup can be attributed to the fact that a substantial amount of time is spent computing the .*? match using individual word and annotation positions. Whereas, the WEAVE indexes do not require such a computation. Second, we consider the speedup over STICH, where we pre-compute the staggered combinations of words and annotations. We observe that WEAVE indexes provide speedups in the range of 4-9× over STICH. This speedup can be attributed to the observation that despite computing the contextual similarity quickly using the 2-stitches, additional time is required to compute the semantic (e.g., *Williams*⊕PERSON) and syntactical similarity (e.g., *Williams* $\xleftarrow{\text{NSUBJ}}$ *won*) using the direct index. Whereas, WEAVE indexes can compute these similarities without the use of the direct index. Third and finally, we consider the speedups over FAST where in addition to the 2-stitch indexes we instantiate 2-fragment indexes that help us quickly compute semantic similarity without the use of direct index. In this case, we observe that WEAVE indexes provide us with a speedup in the range of 5-10×. Here, we also observe that utilizing 2-fragment indexes over the direct index slows down the overall query processing. This is because for many queries verifying the results using the direct index for a smaller result set is

faster than consulting the 2-fragment index whose posting lists can span potentially more documents. Thus, we observe that overall WEAVE indexes provide us with speedups in the range of 4-60× in comparison to the three baselines: WAND, STICH, and FAST.

Results for Reasoning Operators are shown in Table 11, where we report the performance of six reasoning tasks using WEAVE and AGGREGATED WEAVE indexes. First, for the categorization operators, △ and ▽, we observe that the AGGREGATED WEAVE indexes provide an improvement in the ranges of 61-192× and 8-34×, respectively over WEAVE indexes. This speedup can be attributed to the readily summarized triples containing generalizations being stored in the AGGREGATED WEAVE indexes as compared to performing a disjunctive query over pattern sets to obtain the same result set using WEAVE indexes. Moreover, the speedup of ▽ is less than △ because the aggregated lists for the generalization pattern are longer. Second, for the contrastive operators, ↯ and ▷, we observe that the AGGREGATED WEAVE indexes provide speedups in the ranges of 1-2× and 38K- 42K×, respectively over the WEAVE indexes. For the outlier reasoning operator ↯, the performance of AGGREGATED WEAVE and WEAVE indexes are similar as they require the retrieval of direct binary relationship and furthermore are rare in occurrence which results in shorter posting lists. Whereas, for the compare reasoning operator ▷, we obtain a massive improvement as they can often involve reasoning across two to three dependency parse triples. Due to this, the runtimes for resolving the comparative reasoning operator using the WEAVE baseline for C4-NEWS becomes prohibitively expensive at times exceeding 10 hours to resolve hard queries (e.g., $africa \oplus$LOC.). Thus, we can obtain the relative speedup by comparing against the scan baseline — 7K×. Third and finally, for the ranking operators T and N operators we see improvements in the ranges of 7K-19K× and 67K-69K×, respectively over the WEAVE indexes. Similar to the contrastive operators, due to reasoning involving multiple dependency parse triples, we see large improvements between the AGGREGATED WEAVE and WEAVE indexes. In particular, the improvements in numerical reasoning are much larger as the reasoning can often involve more than three hops amongst the dependency parse triples. Thus, like with the comparative reasoning operator, resolving the numerical reasoning operator using the WEAVE indexes becomes infeasible for the largest C4-NEWS collection. For this case, we can compare the relative speedup with respect to the scan baseline — 6K×. Overall, apart from the outlier reasoning operator, we see impressive speedups in the range of 8-69K× across the five remaining reasoning operators using the AGGREGATED WEAVE indexes.

4 Related Work

We have positioned relevant prior-art at appropriate points of discussion in this work. We now describe other related work in relation to search over annotated collections. However, all these approaches can be contrasted to KNITTIR along the following three aspects. First, they do not provide a structured query language that provides similarity search over word sequences or operators for reasoning with noun phrases, named entities, temporal and numerical expressions. Second, they do not provide a presentation that generates vertically partitioned results sets from multiple text regions spread across multiple documents. Third and finally, all these approaches do not scale to large document collections consisting of millions of documents.

Inverted Indexing for Annotated Document Collections. Indicative works in this direction are [13, 17, 23, 26, 27, 29, 30, 58]. Some of the earliest works relied on indexing annotated documents by storing context windows of part-of-speech around a word [17] or the context around named entities [13]. The UIMA framework [23] allows for declarative querying of annotated document collections. The META framework allows text indexing and analysis using topic models [58]. The SAMTLA [29, 30] search system is specifically designed for search tasks in digital humanities. SAMTLA's data model builds upon statistical language models and its storage scheme is implemented using suffix trees. SAMTLA allows users to further access named entity annotations for the retrieved documents and compare documents using edit distance. More recent works, provide GREP-like search over annotated documents [26] and can also structure annotated text into deduplicated tables for a user-defined schema [27].

Graph Indexing for Annotated Document Collections. Representative works for modeling search over annotated document collections by means of graph-based models are [15, 22, 33, 50, 55]. Naïvely, graph-based modeling and search for annotated documents can be done using XML-based XPath query language [33]. The authors in [55] perform XPath based querying that represents dependency parse of sentences as trees and leverages external paraphrase dictionaries to enable similarity search. This is similar in vein to the work by [15], which relies on inverted indexing of parse trees obtained by statistical parsing techniques. Alternatively, [22] models the dependency parse trees as graphs and provide search using a B-Tree based implementation. All these approaches, however, disregard the flexibility that dependency parse trees provide and choose to instead model them as trees. Disregarding parse trees, [50] provide traditional search and ranking at document-level by modeling connections between words, entities, and dates as graphs [50].

Indexing Dense Neural Representations. Using text alone, deep learning models [21, 38, 41] can learn dense representations which are then amenable for similarity search. Recently, [54] attempt to resolve reasoning tasks that leverages these dense representations. However, as discussed in Section 1, vector-based indexes provide limited recall. Furthermore, these dense representations have severe shortcomings in handling temporal and numerical expressions [44] that are central to analytical and reasoning tasks.

5 Conclusion

We described the building blocks for an indexing framework that enables capabilities for transparent, composable, and scalable text analytics. To that end, first we described an analytics-centric search framework that relies on a vertical partitioned result set. Second, we described operators that can retrieve similar text regions and perform reasoning on entity categorization, comparison, and ranking (using temporal and numerical expressions) much like native database operators. Third and finally, we described the design of an indexing infrastructure consisting of weave and aggregated weave indexes. Results show that KNITTIR provides up to 60× factor improvement for retrieval of similar text regions over millions of documents. Furthermore, the reasoning operators can provide augmentations for an entity in the vertically partitioned result set in milliseconds. With KNITTIR, scholars in digital humanities, social scientists, and computational journalists can simplify many of complex analytical tasks over large document collections.

References

[1] Common Crawl. https://commoncrawl.org/the-data/. Accessed: 2024-07-01

[2] ElasticSearch. https://elastic.co/. Accessed: 2024-07-01

[3] English Wikipedia. https://www.wikipedia.org/. Accessed: 2024-07-01

[4] FAISS. https://github.com/facebookresearch/faiss. Accessed: 2024-07-01

[5] JavaFastPFOR: A Simple Integer Compression Library in Java. https://github.com/lemire/JavaFastPFOR. Accessed: 2024-07-01

[6] Lucene. https://lucene.apache.org/. Accessed: 2024-07-01

[7] Neo4J: The Use of Indexes. https://neo4j.com/docs/cypher-manual/current/query-tuning/indexes/. Accessed: 2024-07-01

[8] The New York Times — On This Day. https://learning.blogs.nytimes.com/on-this-day/. Accessed: 2024-07-01

[9] The New York Times Annotated Corpus. https://catalog.ldc.upenn.edu/LDC2008T19. Accessed: 2024-07-01

[10] Understanding Memory and Time Usage. https://stanfordnlp.github.io/CoreNLP/memory-time.html. Accessed: 2024-07-01

[11] Ahmad Issa Alaa Aldine, Mounira Harzallah, Giuseppe Berio, Nicolas Béchet, and Ahmad Faour. 2018. Redefining Hearst Patterns by using Dependency Relations. In *Proceedings of the 10th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management, IC3K 2018, Volume 2: KEOD, Seville, Spain, September 18-20, 2018*, David Aveiro, Jan L. G. Dietz, and Joaquim Filipe (Eds.). SciTePress, 146–153. https://doi.org/10.5220/0006962201460153

[12] Martin Aumüller and Matteo Ceccarello. 2021. The role of local dimensionality measures in benchmarking nearest neighbor search. *Inf. Syst.* 101 (2021), 101807. https://doi.org/10.1016/j.is.2021.101807

[13] Hannah Bast and Björn Buchhold. 2013. An index for efficient semantic full-text search. In *22nd ACM International Conference on Information and Knowledge Management, CIKM'13, San Francisco, CA, USA, October 27 - November 1, 2013*. 369–378. https://doi.org/10.1145/2505515.2505689

[14] Chris Biemann and Martin Riedl. 2013. Text: now in 2D! A framework for lexical expansion with contextual similarity. *J. Lang. Model.* 1, 1 (2013), 55–95. https://doi.org/10.15398/jlm.v1i1.60

[15] Steven Bird, Yi Chen, Susan B. Davidson, Haejoong Lee, and Yifeng Zheng. 2006. Designing and Evaluating an XPath Dialect for Linguistic Queries. In *Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006, 3-8 April 2006, Atlanta, GA, USA*, Ling Liu, Andreas Reuter, Kyu-Young Whang, and Jianjun Zhang (Eds.). IEEE Computer Society, 52. https://doi.org/10.1109/ICDE.2006.48

[16] Alexander Bondarenko, Pavel Braslavski, Michael Völske, Rami Aly, Maik Fröbe, Alexander Panchenko, Chris Biemann, Benno Stein, and Matthias Hagen. 2020. Comparative Web Search Questions. In *WSDM '20: The Thirteenth ACM International Conference on Web Search and Data Mining, Houston, TX, USA, February 3-7, 2020*, James Caverlee, Xia (Ben) Hu, Mounia Lalmas, and Wei Wang (Eds.). ACM, 52–60. https://doi.org/10.1145/3336191.3371848

[17] Michael J. Cafarella and Oren Etzioni. 2005. A search engine for natural language applications. In *Proceedings of the 14th international conference on World Wide Web, WWW 2005, Chiba, Japan, May 10-14, 2005*. 442–452. https://doi.org/10.1145/1060745.1060811

[18] Angel X. Chang and Christopher D. Manning. 2012. SUTime: A library for recognizing and normalizing time expressions. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation, LREC 2012, Istanbul, Turkey, May 23-25, 2012*, Nicoletta Calzolari, Khalid Choukri, Thierry Declerck, Mehmet Ugur Dogan, Bente Maegaard, Joseph Mariani, Jan Odijk, and Stelios Piperidis (Eds.). European Language Resources Association (ELRA), 3735–3740. http://www.lrec-conf.org/proceedings/lrec2012/summaries/284.html

[19] Sarah Cohen, James T. Hamilton, and Fred Turner. 2011. Computational journalism. *Commun. ACM* 54, 10 (2011), 66–71. https://doi.org/10.1145/2001269.2001288

[20] Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. 1990. Indexing by Latent Semantic Analysis. *J. Am. Soc. Inf. Sci.* 41, 6 (1990), 391–407.

[21] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, 4171–4186. https://doi.org/10.18653/v1/n19-1423

[22] T. Krause et al. 2016. graphANNIS: A Fast Query Engine for Deeply Annotated Linguistic Corpora. *Corpus Linguistic Software Tools* 31, 1 (2016), 1–25.

[23] David Ferrucci and Adam Lally. 2004. UIMA: An Architectural Approach to Unstructured Information Processing in the Corporate Research Environment. *Nat. Lang. Eng.* 10, 3-4 (Sept. 2004), 327–348. https://doi.org/10.1017/S1351324904003523

[24] Maayan Geffet and Ido Dagan. 2005. The Distributional Inclusion Hypotheses and Lexical Entailment. In *ACL 2005, 43rd Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, 25-30 June 2005, University of Michigan, USA*, Kevin Knight, Hwee Tou Ng, and Kemal Oflazer (Eds.). The Association for Computer Linguistics, 107–114. https://doi.org/10.3115/1219840.1219854

[25] Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. 2020. Accelerating Large-Scale Inference with Anisotropic Vector Quantization. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event (Proceedings of Machine Learning Research, Vol. 119)*. PMLR, 3887–3896. http://proceedings.mlr.press/v119/guo20h.html

[26] Dhruv Gupta and Klaus Berberich. 2018. GYANI: An Indexing Infrastructure for Knowledge-Centric Tasks. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management, CIKM 2018, Torino, Italy, October 22-26, 2018*, Alfredo Cuzzocrea, James Allan, Norman W. Paton, Divesh Srivastava, Rakesh Agrawal, Andrei Z. Broder, Mohammed J. Zaki, K. Selçuk Candan, Alexandros Labrinidis, Assaf Schuster, and Haixun Wang (Eds.). ACM, 487–496. https://doi.org/10.1145/3269206.3271745

[27] Dhruv Gupta and Klaus Berberich. 2019. JIGSAW: Structuring Text into Tables. In *Proceedings of the 2019 ACM SIGIR International Conference on Theory of Information Retrieval, ICTIR 2019, Santa Clara, CA, USA, October 2-5, 2019*, Yi Fang, Yi Zhang, James Allan, Krisztian Balog, Ben Carterette, and Jiafeng Guo (Eds.). ACM, 237–244. https://doi.org/10.1145/3341981.3344228

[28] Dhruv Gupta and Klaus Berberich. 2020. Optimizing Hyper-Phrase Queries. In *ICTIR '20: The 2020 ACM SIGIR International Conference on the Theory of Information Retrieval, Virtual Event, Norway, September 14-17, 2020*, Krisztian Balog, Vinay Setty, Christina Lioma, Yiqun Liu, Min Zhang, and Klaus Berberich (Eds.). ACM, 41–48. https://doi.org/10.1145/3409256.3409827

[29] Martyn Harris, Mark Levene, Dell Zhang, and Dan Levene. 2014. The anatomy of a search and mining system for digital humanities. In *IEEE/ACM Joint Conference on Digital Libraries, JCDL 2014, London, United Kingdom, September 8-12, 2014*. IEEE Computer Society, 165–168. https://doi.org/10.1109/JCDL.2014.6970163

[30] Martyn Harris, Mark Levene, Dell Zhang, and Dan Levene. 2016. The Anatomy of a Search and Mining System for Digital Archives. *CoRR* abs/1603.07150 (2016). arXiv:1603.07150 http://arxiv.org/abs/1603.07150

[31] Marti A. Hearst. 1992. Automatic Acquisition of Hyponyms from Large Text Corpora. In *14th International Conference on Computational Linguistics, COLING 1992, Nantes, France, August 23-28, 1992*. 539–545. https://aclanthology.org/C92-2082/

[32] Dan Jurafsky and James H. Martin. 2009. *Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition, 2nd Edition*. Prentice Hall, Pearson Education International. https://www.worldcat.org/oclc/315913020

[33] Mounia Lalmas. 2009. *XML Retrieval*. Morgan & Claypool Publishers. https://doi.org/10.2200/S00203ED1V01Y200907ICR007

[34] Rui Liu, Dana McKay, and George Buchanan. 2021. Humanities Scholars and Digital Humanities Projects: Practice Barriers in Tools Usage. In *Linking Theory and Practice of Digital Libraries - 25th International Conference on Theory and Practice of Digital Libraries, TPDL 2021, Virtual Event, September 13-17, 2021, Proceedings (Lecture Notes in Computer Science, Vol. 12866)*, Gerd Berget, Mark Michael Hall, Daniel Brenn, and Sanna Kumpulainen (Eds.). Springer, 215–226. https://doi.org/10.1007/978-3-030-86324-1_25

[35] Shuran Liu and Jun Wang. 2020. How to Organize Digital Tools to Help Scholars in Digital Humanities Research?. In *JCDL '20: Proceedings of the ACM/IEEE Joint Conference on Digital Libraries in 2020, Virtual Event, China, August 1-5, 2020*, Ruhua Huang, Dan Wu, Gary Marchionini, Daqing He, Sally Jo Cunningham, and Preben Hansen (Eds.). ACM, 373–376. https://doi.org/10.1145/3383583.3398615

[36] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to information retrieval*. Cambridge University Press. https://doi.org/10.1017/CBO9780511809071

[37] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP Natural Language Processing Toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*. 55–60. http://www.aclweb.org/anthology/P/P14/P14-5010

[38] Tomás Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems 26:*

27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States, Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger (Eds.). 3111–3119. https://proceedings.neurips.cc/paper/2013/hash/9aa42b31882ec039965f3c4923ce901b-Abstract.html

[39] Marius Muja and David G. Lowe. 2009. Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration. In *VISAPP 2009 - Proceedings of the Fourth International Conference on Computer Vision Theory and Applications, Lisboa, Portugal, February 5-8, 2009 - Volume 1*, Alpesh Ranchordas and Helder Araújo (Eds.). INSTICC Press, 331–340.

[40] Kiril Panev and Klaus Berberich. 2014. Phrase Queries with Inverted + Direct Indexes. In *Web Information Systems Engineering - WISE 2014 - 15th International Conference, Thessaloniki, Greece, October 12-14, 2014, Proceedings, Part I*. 156–169. https://doi.org/10.1007/978-3-319-11749-2_13

[41] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, Alessandro Moschitti, Bo Pang, and Walter Daelemans (Eds.). ACL, 1532–1543. https://doi.org/10.3115/v1/d14-1162

[42] Alina Petrova and Sebastian Rudolph. 2016. Web-Mining Defeasible Knowledge from Concessional Statements. In *Graph-Based Representation and Reasoning - 22nd International Conference on Conceptual Structures, ICCS 2016, Annecy, France, July 5-7, 2016, Proceedings (Lecture Notes in Computer Science, Vol. 9717)*, Ollivier Haemmerlé, Gem Stapleton, and Catherine Faron-Zucker (Eds.). Springer, 191–203. https://doi.org/10.1007/978-3-319-40985-6_15

[43] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2019. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. https://doi.org/10.48550/ARXIV.1910.10683

[44] Anna Rogers, Olga Kovaleva, and Anna Rumshisky. 2020. A Primer in BERTology: What We Know About How BERT Works. *Trans. Assoc. Comput. Linguistics* 8 (2020), 842–866. https://transacl.org/ojs/index.php/tacl/article/view/2257

[45] Stephen Roller, Douwe Kiela, and Maximilian Nickel. 2018. Hearst Patterns Revisited: Automatic Hypernym Detection from Large Text Corpora. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 2: Short Papers*, Iryna Gurevych and Yusuke Miyao (Eds.). Association for Computational Linguistics, 358–363. https://doi.org/10.18653/V1/P18-2057

[46] Erik F. Tjong Kim Sang and Katja Hofmann. 2009. Lexical Patterns or Dependency Patterns: Which Is Better for Hypernym Extraction?. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning, CoNLL 2009, Boulder, Colorado, USA, June 4-5, 2009*, Suzanne Stevenson and Xavier Carreras (Eds.). ACL, 174–182. https://aclanthology.org/W09-1122/

[47] Sebastian Schuster and Christopher D. Manning. 2016. Enhanced English Universal Dependencies: An Improved Representation for Natural Language Understanding Tasks. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation LREC 2016, Portorož, Slovenia, May 23-28, 2016*, Nicoletta Calzolari, Khalid Choukri, Thierry Declerck, Sara Goggi, Marko Grobelnik, Bente Maegaard, Joseph Mariani, Hélène Mazo, Asunción Moreno, Jan Odijk, and Stelios Piperidis (Eds.). European Language Resources Association (ELRA). http://www.lrec-conf.org/proceedings/lrec2016/summaries/779.html

[48] Julian Seitner, Christian Bizer, Kai Eckert, Stefano Faralli, Robert Meusel, Heiko Paulheim, and Simone Paolo Ponzetto. 2016. A Large DataBase of Hypernymy Relations Extracted from the Web. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation LREC 2016, Portorož, Slovenia, May 23-28, 2016*, Nicoletta Calzolari, Khalid Choukri, Thierry Declerck, Sara Goggi, Marko Grobelnik, Bente Maegaard, Joseph Mariani, Hélène Mazo, Asunción Moreno, Jan Odijk, and Stelios Piperidis (Eds.). European Language Resources Association (ELRA). http://www.lrec-conf.org/proceedings/lrec2016/summaries/204.html

[49] Rion Snow, Daniel Jurafsky, and Andrew Y. Ng. 2004. Learning Syntactic Patterns for Automatic Hypernym Discovery. In *Advances in Neural Information Processing Systems 17 [Neural Information Processing Systems, NIPS 2004, December 13-18, 2004, Vancouver, British Columbia, Canada]*. 1297–1304. https://proceedings.neurips.cc/paper/2004/hash/358aee4cc897452c00244351e4d91f69-Abstract.html

[50] Andreas Spitz and Michael Gertz. 2016. Terms over LOAD: Leveraging Named Entities for Cross-Document Extraction and Summarization of Events. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval, SIGIR 2016, Pisa, Italy, July 17-21, 2016*, Raffaele Perego, Fabrizio Sebastiani, Javed A. Aslam, Ian Ruthven, and Justin Zobel (Eds.). ACM, 503–512. https://doi.org/10.1145/2911451.2911529

[51] Maite Taboada and María de los Ángeles Gómez-González. 2012. Discourse markers and coherence relations: Comparison across markers, languages and modalities. *Linguistics and the Human Sciences* 6, 1-3 (Dec. 2012), 17–41. https://doi.org/10.1558/lhs.v6i1-3.17

[52] Melissa Terras, James Baker, James Hetherington, David Beavan, Martin Zaltz Austwick, Anne Welsh, Helen O'Neill, Will Finley, Oliver Duke-Williams, and Adam Farquhar. 2018. Enabling complex analysis of large-scale digital collections: humanities research, high-performance computing, and transforming access to British Library digital collections. *Digit. Scholarsh. Humanit.* 33, 2 (2018), 456–466. https://doi.org/10.1093/LLC/FQX020

[53] Stefan Thater, Hagen Fürstenau, and Manfred Pinkal. 2010. Contextualizing Semantic Representations Using Syntactically Enriched Vector Models. In *ACL 2010, Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, July 11-16, 2010, Uppsala, Sweden*, Jan Hajic, Sandra Carberry, and Stephen Clark (Eds.). The Association for Computer Linguistics, 948–957. https://aclanthology.org/P10-1097/

[54] James Thorne, Majid Yazdani, Marzieh Saeidi, Fabrizio Silvestri, Sebastian Riedel, and Alon Y. Halevy. 2021. Database reasoning over text. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (Eds.). Association for Computational Linguistics, 3091–3104. https://doi.org/10.18653/v1/2021.acl-long.241

[55] Xiaolan Wang, Aaron Feng, Behzad Golshan, Alon Y. Halevy, George A. Mihaila, Hidekazu Oiwa, and Wang-Chiew Tan. 2018. Scalable Semantic Querying of Text. *Proc. VLDB Endow.* 11, 9 (2018), 961–974. https://doi.org/10.14778/3213880.3213887

[56] Hugh E. Williams, Justin Zobel, and Dirk Bahle. 2004. Fast Phrase Querying with Combined Indexes. *ACM Trans. Inf. Syst.* 22, 4 (Oct. 2004), 573–594. https://doi.org/10.1145/1028099.1028102

[57] Xifeng Yan, Philip S. Yu, and Jiawei Han. 2004. Graph Indexing: A Frequent Structure-based Approach. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Paris, France, June 13-18, 2004*, Gerhard Weikum, Arnd Christian König, and Stefan Deßloch (Eds.). ACM, 335–346. https://doi.org/10.1145/1007568.1007607

[58] ChengXiang Zhai and Sean Massung. 2016. *Text Data Management and Analysis: A Practical Introduction to Information Retrieval and Text Mining.* Association for Computing Machinery and Morgan & Claypool, New York, NY, USA.

[59] Marcin Zukowski, Sandor Heman, Niels Nes, and Peter Boncz. 2006. Super-Scalar RAM-CPU Cache Compression. In *Proceedings of the 22Nd International Conference on Data Engineering (ICDE '06)*. IEEE Computer Society, Washington, DC, USA, 59–. https://doi.org/10.1109/ICDE.2006.150