

# Online Annotation of Text Streams With Structured Entities

Ken Q. Pu  
UOIT  
Oshawa, Canada  
ken.pu@uoit.ca

Okkie Hassanzadeh  
University of Toronto  
Toronto, Canada  
okkie@cs.toronto.edu

Richard Drake  
UOIT  
Oshawa, Canada  
richard.drake@uoit.ca

Renée J. Miller  
University of Toronto  
Toronto, Canada  
miller@cs.toronto.edu

## ABSTRACT

We propose a framework and algorithm for annotating unbounded text streams with entities of a structured database. The algorithm allows one to correlate unstructured and dirty text streams from sources such as emails, chats and blogs, to entities stored in structured databases. In contrast to previous work on entity extraction, our emphasis is on performing entity annotation in a completely online fashion. The algorithm continuously extracts important phrases and assigns to them top- $k$  relevant entities. Our algorithm does so with a guarantee of constant time and space complexity for each additional word in the text stream, thus infinite text streams can be annotated. Our framework allows the online annotation algorithm to adapt to changing stream rate by self-adjusting multiple run-time parameters to reduce or improve the quality of annotation for fast or slow streams, respectively. The framework also allows the online annotation algorithm to incorporate query feedback to learn user preferences and personalize the annotation for individual users.

## Categories and Subject Descriptors

H.3.3 [Information Storage and Retrieval]: Information Search and Retrieval

## General Terms

Algorithms, Performance

## 1. INTRODUCTION: ADAPTIVE ONLINE ENTITY ANNOTATION

Information comes in many forms including unstructured text and structured databases. In recent years, there is a growing interest in combining these two forms of information. For example, the current Web is often considered a web of (unstructured) documents, and many people have put forth the vision that it should become a web of structured data with the semantics this entails. An important

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

CIKM'10, October 26–30, 2010, Toronto, Ontario, Canada  
Copyright 2010 ACM 978-1-4503-0099-5/10/10 ...\$10.00.

approach to achieving this vision is to identify linkages between parts (phrases) of text data with entities in structured databases [1, 2, 3, 13, 14]. With the popularity of searching information with keyword queries, one may argue that the volume of textual information is growing much more rapidly than structured data. Furthermore, with the rising popularity in real-time collaborative communication technology (e.g., Google Wave) and text messaging from mobile devices, much of this unstructured data must be processed in its original streaming form. This motivates us to investigate the problem of online entity extraction over (potentially infinite) streams. For an entity extraction algorithm to be *online*, it must exhibit constant time and space complexity to process each new word in the text stream.

Text streams come at different rates. An active blog account has  $\approx 1$  blog per day, and each blog may contain  $\approx 5000$  words. So the text rate is only 0.05 words per second. An email account may receive  $\approx 100$  email messages per day, and with  $\approx 500$  words per email, we have a text rate of 0.5 words per second. In an interactive environment, a typical user can type at 0.5 to 1 word per second. In these scenarios, the text rate is slow, so a real-time entity extraction algorithm should be able to exploit sophisticated text and entity matching algorithms to achieve high information retrieval accuracy. On the other hand, if the entity extraction algorithm is to be deployed at an email server that is supporting 1000 email accounts, then the algorithm must deal with streams of email messages at a text rate of 500 words per second. It is desirable for the extraction algorithm to be able to self-adapt to the high text rate by reducing the degree of complexity of text-entity matching. So, in addition to the properties of an online algorithm, such an approach must be able to automatically adjust the complexity of the text analysis to accommodate for variable rates of the text stream.

Finally, the entity extraction algorithm should be adaptive to individual user preferences. For email streams, the discussion of a marketing manager may be highly specific to database entities that are relevant to marketing rather than say human resources. Thus, the search algorithm should *learn* and incorporate such user preferences.

The main contribution of this paper is an algorithm that annotates (nondeterministically) a text stream with the relevant entities. The algorithm exhibits the following properties.

- Complexity of incremental annotation for each additional text word is constant, in both space and time.
- The annotation algorithm supports fuzzy string matching.

- The annotation algorithm can be tuned to exploit the trade-off between precision and performance with a set of runtime parameters.
- When query feedback is available, the annotation algorithm learns the user preference and uses it to improve the accuracy of the annotations.

We begin with a discussion of the related work on which we are building and identify the key new contributions of our approach. The formal definition of entity annotation over text streams is given in Section 3. The annotation algorithm and its important properties are presented in detail in Section 4. Section 5 presents methods of self-tuning to the run-time environment (text rate) and personalized search based on learned user preferences. We have tested our approach using several representative data sets. The experimental evaluations are presented in Section 6. Future extensions to this paper are outlined in Section 7.

## 2. RELATED WORK

Entity extraction or recognition is the problem of identifying entities from unstructured text. Recently, there has been an increasing interest in techniques that exploit the existence of a structured database (or dictionary) of entities to improve the accuracy and performance of the entity recognition [1, 2, 3, 4, 14]. Some of these approaches, like ours, assume an accurate database of entities, but accommodate errors in the unstructured text [2, 14]. However, to the best of our knowledge, no entity extraction techniques provide an online solution with constant time and space complexity.

Entity recognition is also closely related to the database problem of providing keyword search over a structured or semi-structured database [16]. In such work, an (unstructured) keyword query is “matched” to entities in a structured database, or used to identify tuples in multiple tables that may be relevant to the query. Some approaches assume clean, unambiguous queries that refer to a single database entity. Closer to our work are approaches that segment the query (keywords) to provide better entity matching to multiple database entities [8]. However, dirty queries containing spelling errors or syntactic differences are generally not handled well by such approaches. To address this, recent work has considered the problem of keyword segmentation in the presence of spelling errors and semantic mis-matches and the problem of doing incremental segmentation of the (unstructured) query [12]. Both features are similar to and inspire our work in that we consider dirty unstructured text and incremental segmentation. Notably, our approach considers text that is an infinite stream, not a small keyword query. Furthermore, we present an adaptive online algorithm that can adjust to variable text streaming rates and also to user preferences for certain entities over others.

Work in web search and information retrieval has also addressed several problems related to segmenting queries or longer pieces of text. Examples include Query by Document (QBD) [15], and Yahoo! Term Extraction API which is part of Yahoo! Search Web Services<sup>1</sup>. Like our work, Yahoo! Term Extraction extracts (segments) key phrases from text. We augment this capability by considering the relevance of a segmented phrase to the entities in a structure database and use this information to achieve a better segmentation. The QBD approach also does segmentation independent of

<sup>1</sup><http://developer.yahoo.com/search/>

the database, but then ranks the segments by querying the target data (where the end goal is quite different from ours - to rank related documents). The TASTIER<sup>2</sup> [6] and CompleteSearch<sup>3</sup> (or ESTER) systems use efficient algorithms for type-ahead entity matching. Both systems show the partially matched results of a (partial) keyword query (as it is being typed) in an online fashion. TASTIER also performs fuzzy string matching to capture spelling errors. Our approach is complementary as it performs online segmentation in addition to approximate entity matching. Unlike TASTIER and CompleteSearch, our system finds entities in the input text that are not necessarily associated with each other. Furthermore, our system can automatically adapt to text streams with highly variable rates.

Our work is also related to the active area of semantic annotation in knowledge management (KM). Uren et al. [13] present a survey of existing manual and (semi-)automatic annotation and entity extraction techniques. As an example, KnowItAll [5] performs unsupervised named-entity extraction from the Web, based on specification of patterns by the user, e.g., extracting city names by finding occurrences of the phrase “cities such as ...” in web documents. AktiveDoc<sup>4</sup> performs annotation on-the-fly while reading or editing documents, but, to the best of our knowledge, does not allow any errors. Within the document-centric knowledge management model presented by Uren et al. [13], our work can be seen as a novel fully automatic (unsupervised) and online annotation technique that can be added to existing KM systems to allow a different type of annotation. This annotation does not require supervised learning or specification of rules or patterns, and can perform annotation on the fly, even in the presence of syntactic differences (between the streamed text and database) or spelling errors.

## 3. MOTIVATION, DEFINITIONS AND PROBLEM STATEMENT

Consider a user who is engaging in an online chat involving the following text:

```
... doctor pepper mint tea watching few good
men with peter allen ...
```

Let us also imagine that one has access to several databases of structured information containing among other things information on food products, movies and actors. It would be desirable to highlight parts of the text stream with information about relevant entities. For instance, we may wish to augment the stream with the following annotations (in some graphical interface):

‘doctor pepper’	Soft Drink: 150 Calories, Introduced in 1885 by Charles Alderton
‘mint tea’	Herbal Drink: 0 Calories
‘few good men’	Movie: “FEW GOOD MEN, A”, 1992
‘peter allen’	Actor: “ALLEN, PETER”

However, this is only one possible set of annotations. More possibilities arise if we consider approximate string matching. For instance, it may be reasonable to annotate the phrase ‘peter allen’ with the entity Actor: “ALAN, PETER”. Non-determinism also exists in the way the stream is

<sup>2</sup><http://tastier.ics.uci.edu/>

<sup>3</sup><http://dblp.mpi-inf.mpg.de/>

<sup>4</sup><http://nlp.shef.ac.uk/wig/aktivedoc.htm>

segmented into phrases. One possibility is to group ‘pepper mint’ together as a segment or phrase, and annotate it as a spice entity, and to group ‘tea’ by itself, and annotate it with various hot beverage entities of different types of tea. The possible phrases and entity annotations should be scored to reflect their respective quality and ranked by these scores. Our objective is to generate a ranked set of annotated phrases for continuously streamed text. The annotation algorithm must exhibit constant time and space complexity so text streams with unbounded length can be handled.

We begin with a set of basic definitions. We then formally define the problem of online entity annotation.

### 3.1 Basic definitions

A text stream,  $T$ , is a (possibly infinite) sequence of words  $w_0, w_1, \dots, w_i, \dots$ . A range,  $T(i, j)$ , in the stream is a finite sequence of consecutive words from position  $i$  inclusively, to position  $j$  exclusively. When there is no risk of confusion, we may treat  $T(i, j)$  both as a sequence of words, or a single string obtained by joining words  $T(i, j)$  by spaces.

For the purpose of this paper, an entity is characterized by a tuple  $e = (x_{\text{id}}, x_{\text{type}}, x_{\text{text}})$  where  $x_{\text{id}}$  is the unique identifier of the entity,  $x_{\text{type}}$  is the type or schema of the entity, and  $x_{\text{text}}$  is the text value associated with the entity. We use the notation  $\text{type}(e) = x_{\text{type}}$  and  $\text{text}(e) = x_{\text{text}}$ . The set of all possible entities is denoted by  $\mathbf{E}$ .

One can enrich the definition with additional attributes and data types. However, for the purpose of entity extraction, we are only interested in the type and text value associated with the entity as these will be used to match the entity to the text stream. Additional descriptive information about an entity can be retrieved from the database using its entity identifier.

A phrase  $p$  is a range  $T(i, j)$  in the text stream that may match an entity. We denote all possible phrases in a text stream  $T$  by  $\mathbf{Phrase}(T)$ . From this point on, we distinguish phrases from ranges by using lower case letters  $i, j, k, \dots$  to denote boundaries and indices of phrases, and upper case letters  $I, J, K, \dots$  to denote boundaries and indices of ranges.

An entity annotation of a phrase  $p$  is an assignment  $p \mapsto e$  that maps the phrase  $p$  to some entity  $e$ . In order to support approximate matching and top- $k$  search, we generalize entity annotations to support non-deterministic annotations of a phrase.

**DEFINITION 1 (SCORED SETS).** *Let  $X$  be a set. A scored subset of  $X$  is some subset  $Y \subseteq X$ , together with a scoring function  $\text{score}_Y : Y \rightarrow \mathbb{R}^+$ . The collection of all possible scored subsets of  $X$  is denoted by  $\mathcal{P}_{\text{score}}(X)$ .*

**DEFINITION 2 (NON-DETERMINISTIC ENTITY ANNOTATION).** *Given a phrase  $p$  in a text stream  $T$ , a non-deterministic entity annotation of  $p$  is a mapping  $p \mapsto E$  where  $E \in \mathcal{P}_{\text{score}}(\mathbf{E})$ .*

A non-deterministic entity annotation  $p \mapsto E$  implies that the phrase  $p$  corresponds to one of the entities in  $E$ . The score  $\text{score}_E(e)$  for  $e \in E$  indicates the similarity between the contents of the phrase and the text value ( $\text{text}(e)$ ) of the entity.

The entity annotation problem involves identifying phrases in the text stream that can be annotated by entities. A *phrase assignment* is a function that identifies interesting phrases.

**DEFINITION 3 (PHRASE ASSIGNMENT).** *Given a text stream,  $T$ , a phrase assignment,  $P$ , is a partial function:*

$$P : \mathbb{N} \rightarrow \mathbf{Phrase}(T) \cup \{\perp\}$$

*such that, for all  $k \in \mathbb{N}$ , either  $P(k)$  is undefined, or  $P(k) = \perp$ , or  $P(k) = T(i, j)$ , where  $i \leq k < j$ . Furthermore,*

$$\forall k, P(k) = T(i, j) \in \mathbf{Phrase}(T) \implies \forall k' \in [i, j), P(k') = P(k)$$

*Given a phrase  $p \in \mathbf{Phrase}(T)$ , we say  $p \in P$  if  $\exists k \in \mathbb{N}, P(k) = p$ .*

*A phrase assignment,  $P$ , is annotated if there exists a phrase annotation function  $h$  such that  $p \in P \implies h(p) \in \mathbf{E}$ . The phrase assignment  $P$  is non-deterministically annotated if  $h(p) \in \mathcal{P}_{\text{score}}(\mathbf{E})$  for all  $p \in P$ .*

Intuitively, a phrase assignment labels each word in the stream with the phrases that it belongs to. Given a word  $w_k$ , if  $P(k) = T(i, j)$ , it implies that the word  $w_k$  is part of a phrase  $T(i, j)$ , and hence the condition of  $i \leq k < j$ . Since the range  $T(i, j)$  is grouped into a phrase, all positions  $k \in \mathbb{N}$ , such that  $i \leq k < j$ , must be assigned to the phrase  $T(i, j)$ . Finally,  $P(k) = \perp$  indicates that the word  $w_k$  does not belong to any phrase. Note that phrase assignments may be partial functions, so they may only label parts of the text stream. Note,  $P(k)$  being undefined and  $P(k) = \perp$  are not equivalent. We refer to the number of phrases  $p$  where  $p \in P$  as the length of  $P$ , written  $\text{length}(P)$ . An annotated phrase assignment is a phrase assignment whose phrases are annotated by entities.

Generally, given a range,  $T(I, J)$ , there are different phrase assignments that make sense, and for each phrase, there are many different candidate entities that can be used for annotation. The non-determinism of both the phrase assignment and the entity annotation over a range  $T(I, J)$  is captured by the notation of a frame over  $(I, J)$ .

**DEFINITION 4 (FRAMES).** *A frame  $M$  is a collection of non-deterministically annotated phrase assignments over a range  $T(I, J)$ . So each  $P \in M$  is defined over the domain  $(I, J)$ , and their phrases are annotated by scored sets of entities.*

We write  $M(I, J)$  to indicate that  $M$  is a frame over the range  $T(I, J)$ . A frame is a subset of the possible segmentations. The rest of the paper discusses the scoring and generation of frames from a text stream.

**EXAMPLE 1.** *Consider a range of a text stream from position 10 to position 13:*

... 10      11      12      13    ...  
... doctor pepper mint    tea ...

*The phrase  $p = T(11, 13)$  is the range containing the words: ‘pepper’, ‘mint’. An instance of a non-deterministic entity annotation of  $p$  is the mapping  $h$ :*

$$p \xrightarrow{h} \left\{ \left[ \begin{array}{l} \text{type : FOOD/MISC} \\ \text{text : 'pepper mint gum'} \end{array} \right], \left[ \begin{array}{l} \text{type : FOOD/SNACK} \\ \text{text : 'pepper mint cookies'} \end{array} \right] \right\}$$

*We omitted the entity identifiers, as well as the score assigned to each entity. A phrase assignment over the range  $T(10, 14)$  is a mapping,  $P$ , of each word in the range, such as the following.*

$w_k$	doctor	pepper	mint	tea
$P(k)$	$\perp$	$T(11, 13)$	$T(11, 13)$	$T(13, 14)$

According to the phrase assignment  $P$ , ‘pepper’ and ‘mint’ are grouped together into one phrase, ‘tea’ is grouped into a phrase by itself, and the word ‘doctor’ is not mapped to any entity. The phrase assignment  $P$  together with the annotation  $h$  forms an annotated phrase assignment.

A frame  $M(10, 14)$  over the range consists of multiple phrase assignments. Below is an instance of a phrase assignment.

$$M = \begin{array}{|c|c|c|} \hline \text{‘doctor’} & \text{‘pepper mint’} & \text{‘tea’} \\ \hline \text{‘doctor pepper’} & \text{‘mint tea’} & \\ \hline \text{‘doctor’} & \text{‘pepper mint tea’} & \\ \hline \text{‘doctor pepper’} & \text{‘mint’} & \text{‘tea’} \\ \hline \end{array}$$

Each phrase in the assignments in  $M$  is further annotated by some non-deterministic entity annotation function. Note that the phrases in an assignment need not be contiguous.

In light of the non-determinism in entity annotation of phrases and phrase assignments over the same range, we need to rank the different possibilities by a scoring function. We use a similarity function  $\text{SIMILARITY} : \text{String} \times \text{String} \rightarrow \mathbb{R}^+$ .<sup>5</sup>

**DEFINITION 5 (SCORES OF PHRASE AND ASSIGNMENT).** The scoring function for phrases is defined as:

$$\text{score} : \mathbf{Phrase}(T) \rightarrow \mathbb{R}^+ : p \mapsto \max_{e \in \mathbf{E}} \text{SIMILARITY}(p, \text{text}(e))$$

The score for a phrase assignment  $P$  is the average score of all the phrases in  $P$ :  $\text{score}(P) = \sum_{p \in P} \text{score}(p) / \text{length}(P)$ .

For  $p = T(i, j)$ , we write  $\text{score}(T(i, j))$  as  $\text{score}(i, j)$  for brevity. For the rest of the paper, whenever we talk about a scored set  $X$  of phrases or phrase assignments,  $\text{score}_X$  is assumed to be the scoring function for phrases or phrase assignments, respectively.

### 3.2 The Problem

**DEFINITION 6 (ADMISSIBLE PHRASES).** Let  $c > 0$  be some constant value in  $\mathbb{R}^+$ . We say that a phrase  $p$  is  $c$ -admissible if  $\text{score}(p) \geq c$ . Given a range  $T(I, J)$ , we use  $\mathcal{A}_c(I, J)$  to denote all  $c$ -admissible phrases in the range. We say that  $c$  is the admissibility threshold, written  $C_{\text{closed}}$ .

The term *closed* refers to the fact that words in a phrase are compared to the entity text without the possibility of being extended further. In contrast, we later define a notion of *open similarity* comparison.

**DEFINITION 7 (MAXIMAL PHRASE ASSIGNMENTS).** Let  $X$  be a collection of phrases. A phrase assignment  $P$  is  $X$ -maximal if all phrases in  $P$  belong to  $X$ , and further more,

$$\forall p \in X, p \notin P \implies \exists p' \in P, p' \cap p \neq \emptyset$$

An  $X$ -maximal phrase assignment  $P$  asserts that  $P$  cannot be augmented by any phrases in  $X$  and still remain a valid phrase assignment.

**PROBLEM 1 (ONLINE OPTIMAL PHRASE ASSIGNMENT).** The online optimal phrase assignment problem is defined as:

- **Input:** A text stream,  $T$ , and an admissibility threshold,  $C_{\text{closed}}$ .

<sup>5</sup>Further discussion of the similarity function is deferred to later sections.

- **Output:** For each  $I \in \mathbb{N}$ , compute the  $\mathcal{A}_{C_{\text{closed}}}(1, I)$ -maximal phrase assignment that has the greatest score. Denote the optimal phrase assignment as  $P_I^*$ .

The  $\mathcal{A}_{C_{\text{closed}}}(1, I)$ -maximality requirement in Problem 1 enforces that each phrase in the assignment is  $C_{\text{closed}}$  admissible, and that no other admissible phrases can be added to the phrase assignment.

In order to handle infinite streams, in practice, we further impose the constraint that for each time step  $I$ ,  $P_I^*$  can be computed with the time and space complexity of  $\mathcal{O}(1)$  with respect to  $I$ . A keen reader may see something troublesome: the number of phrases,  $\text{length}(P_I^*)$ , increases as  $I$  increases, so it is not guaranteed that it is possible at all to compute  $P_I^*$  in constant time in general. There is, yet, another problem with the definition of Problem 1.

In dealing with the non-determinism of phrase assignments, one may be tempted to simply generalize Problem 1 to an online top- $k$  phrase assignments problem. However, the top- $k$  analogue to Problem 1 has very little value in practice. Let us demonstrate by an example.

**EXAMPLE 2.** Consider a text stream containing the following range: ‘... doctor pepper mint tea ... programming java coffee ...’

Individually we have the following phrase assignments ranked by their scores.

doctor	pepper mint tea	1.0	programming java	coffee	0.9
doctor pepper	mint tea	0.9	programming	java coffee	0.75
doctor	pepper mint	0.9			

The top-3 phrase assignments for this range of the stream consists of:

doctor	pepper mint tea	programming java	coffee
doctor pepper	mint tea	programming java	coffee
doctor	pepper mint	tea	programming java

Something very alarming can be seen: the possibility of ‘programming’ ‘java coffee’ is eliminated from the top-3 list. Increasing the  $k$  value for the top- $k$  phrase assignments will not solve the problem in a fundamental way.

As the stream grows, more “independent” ranges form, and the total number of candidate phrase assignments of the whole stream is the product of the phrase assignments of the individual ranges. So, as the stream grows, the total search space grows exponentially. If only top- $k$  of the *whole* stream is to be computed, then the search result becomes diminishing small, and more and more interesting phrase assignments will be lost, as demonstrated in Example 2.

The solution is to report the top- $k$  phrase assignments for each sub-range. In Example 2, we see two distinct ranges: ‘doctor pepper mint tea’ and ‘programming java coffee’. So, we need to redefine the optimal phrase assignment problem in such a way that the algorithm computes the individual regions, and the top- $k$  phrase assignments for each region.

**DEFINITION 8 (STABLE RANGES).** Given a collection  $X$  of phrases, the projection  $\pi_{I, J}(X)$  is defined as,

$$\pi_{I, J}(X) = \{T(i, j) \in X : i \in [I, J] \text{ or } j \in [I, J]\}$$

Let  $c$  be the admissibility threshold. A range  $T(I, J)$  is  $c$ -stable if for all  $I' \leq I$  and  $J' \geq J$ , we have  $\mathcal{A}_c(I, J) = \pi_{I, J}(\mathcal{A}_c(I', J'))$ . The range is minimally stable if  $(I + 1, J)$  and  $(I, J - 1)$  are not stable. The set of all minimally stable ranges is denoted by  $\mathcal{M}(T)$ .

The significance of a stable range is that its set of admissible phrases is unaffected by what comes before and after it in the text stream.

EXAMPLE 3. Consider the range: ‘having pepper mint tea programming java coffee all the time’. Let the entity database contain entities with text values: ‘doctor pepper’, ‘java coffee’, ‘mint tea’, ‘programming java’. Let  $T(I, J) = \text{‘pepper mint tea’}$ . The range  $T(I, J)$  is stable because phrases  $A(I, J) = \{\text{‘pepper mint’}, \text{‘mint tea’}\}$  which can be always obtained from  $A(I', J')$  where  $(I, J) \subseteq (I', J')$  by means of projection back to  $(I, J)$ . On the other hand ‘pepper mint’ is not stable because its phrases cannot be obtained by projecting  $A(I, J)$ :

$$\begin{aligned} A(\text{‘pepper mint’}) &= \{\text{‘pepper mint’}\} \\ A(\text{‘pepper mint tea’}) &= \{\text{‘pepper mint’}, \text{‘mint tea’}\} \\ \pi_{\text{‘pepper mint’}}(A(\text{‘pepper mint tea’})) \\ &= \{\text{‘pepper mint’}, \text{‘mint tea’}\} \\ &\neq A(\text{‘pepper mint’}) \end{aligned}$$

The following properties of minimally stable ranges follow immediately from the definitions.

PROPOSITION 3.1. All distinct minimally stable ranges of  $T$  are disjoint. Consequently,  $\mathcal{M}(T)$  is totally ordered.

PROPOSITION 3.2 (BOUNDARIES OF STABLE RANGES). Given a position  $I$  in the stream. If for all  $T(i, j) \in \mathcal{A}_c(T)$ ,  $I \notin (i, j)$ , then  $I$  is a boundary of two stable ranges.

Proposition 3.1 makes it possible to define the last minimally stable range before  $I$  for any position  $I$ . The last minimally stable range before  $I$  is given by:

$$\max\{T(J, K) \in \mathcal{M} : K \leq I\}$$

We generalize Problem 1 by always computing the top- $k$  annotated phrase assignment for the last stable range.

PROBLEM 2 (ONLINE TOP- $k$  PHRASE ASSIGNMENT). The online top- $k$  phrase assignment over stable ranges is defined as:

- **Input:** a text stream,  $T$ , and an admissibility threshold  $C_{\text{closed}}$ .
- **Output:** for each position  $I$ , compute the last minimally stable range  $M_i$  (w.r.t.  $\mathcal{A}_{C_{\text{closed}}}$ ) and its top- $k$  phrase assignments over  $M_i$ .

By Definition 4, the solution to the problem is a stream of frames ranging over minimally stable ranges, and containing the top- $k$  optimal phrase assignments ranked according to the scoring function defined in Definition 5. In the case of top-1, Problem 2 computes  $P_I^*$  in a streamed fashion: it yields a stream of segments of  $P_I^*$ .

### 3.3 Open similarity

The score of a phrase indicates how well it matches entities in the database. However, we are interested in an online solution that permits us to score phrases as they are being presented in a stream, even if they are not yet complete. This is particularly important for the case of online annotation, for we do not have the entire text stream to analyze. To do this, we use an additional score  $\text{score}_o(p)$  to reflect how well a phrase  $p$  partially matches entities in the database. We refer to  $\text{score}_o(p)$  as the open-score.

DEFINITION 9 (OPEN SIMILARITY AND OPEN SCORES). Let  $p$  be a phrase, and  $e \in \mathbf{E}$  an entity. Let  $\mathbf{Words}(p)$  and  $\mathbf{Words}(e)$  be, respectively, the words in the phrase and words in the entity text value (split by white space). Define a bipartite graph  $G$  with weighted edges as:

- Nodes,  $V(G)$ , of  $G$  are words in  $p$  and  $\text{text}(e)$ .
- Edges,  $E(G)$ , of  $G$  are  $\mathbf{Words}(p) \times \mathbf{Words}(e)$ .
- The weight of an edge  $(w, w')$  is the similarity measure  $\text{SIMILARITY}(w, w')$ .

Let  $M \subseteq E(G)$  be the maximal matching in  $G$ . The open similarity  $\text{SIMILARITY}_o(p, e)$  is given as:

$$\text{SIMILARITY}_o(p, e) = \text{average}\{\text{SIMILARITY}(w, w') : (w, w') \in M\}$$

The open score of  $p$  is the best possible open similarity:

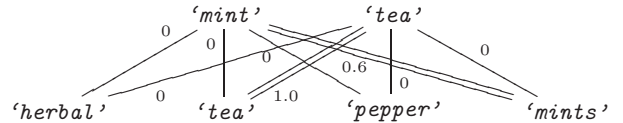
$$\text{score}_o(p) = \max_{e \in \mathbf{E}} \text{SIMILARITY}_o(p, e)$$

In Definition 9, we assume that  $|\mathbf{Words}(p)| = |\mathbf{Words}(e)|$  without loss of generality, for one can always pad either  $\mathbf{Words}(p)$  or  $\mathbf{Words}(e)$  with slack words and set  $\text{SIMILARITY}(w, w') = 0$  if  $w$  or  $w'$  is a slack word.

EXAMPLE 4. Consider the phrase  $p = \langle \text{‘mint’}, \text{‘tea’} \rangle$ , and the entity  $e = \begin{bmatrix} \text{type:} & \text{HOT DRINKS} \\ \text{text:} & \text{‘herbal tea: pepper mints’} \end{bmatrix}$

Clearly, the two should have a high similarity measure (as it is possible that  $p$  is extended into the entity text). However, if 3-gram is used, then the Jaccard similarity between  $p$  and  $\text{text}(e)$  is only 0.28. The open similarity reflects a very different story.

The graph  $G$  as in Definition 9 is as follows.<sup>6</sup>



The nodes are the words in  $p$  and the text value of  $e$ . The weights on the edges are the Jaccard similarity between 3-grams of the words. The maximum matching is the edges with double lines.

$$\text{SIMILARITY}_o(p, e) = \text{average}\{0.6, 1.0\} = 0.8$$

We remark that the open similarity  $\text{SIMILARITY}_o(p, e)$  can be computed exactly in  $\mathcal{O}(n^3)$  where  $n$  is the total word count in  $p$  and  $\text{text}(e)$ , using the Hungarian algorithm [7].

## 4. THE ALGORITHMIC SOLUTION

Before presenting an exact algorithm, we need to provide definitions on how phrases and phrase assignments can be extended.

DEFINITION 10 (PHRASE EXTENSION WITH THRESHOLD). Let  $p = T(i, j)$  be a phrase, and  $w_j$  be the word immediately after  $p$  in the text stream. Let  $C_{\text{open}} > 0$  be a constant. Denote

$$\mathbf{Good}_o(i, j) \equiv \text{score}_o(i, j) \geq C_{\text{open}}$$

<sup>6</sup>When constructing the set of words of a string, we use the standard practice of removing non-alphanumerics (e.g. ‘:’), and splitting by whitespace.

---

**Algorithm 1** All phrase assignments over stable ranges.

---

```
1:  $M \leftarrow \{\emptyset\}$   $\{M$  is a frame $\}$ 
2: for every position  $i$  in stream  $T$  do
3:   try:  $M' = \bigcup_{P \in M} P \oplus w_i$ 
4:   on success:
5:      $M \leftarrow M'$ 
6:   on error: Not mergable
7:     yield  $\text{top}_k(M)$ 
8:     if  $\text{Good}_o(i, i+1)$  then
9:        $M = \{(i, i+1)\}$ 
10:    else
11:       $M = \{\emptyset\}$ 
12:    end if
13: end for
```

---

$$\text{Good}_c(i, j) \equiv \text{score}(i, j) \geq C_{\text{closed}}$$

The non-deterministic phrase extension  $p \oplus w_j$  is defined as:

$$\begin{aligned} p \oplus w_j = & \text{if } \text{Good}_o(i, j+1) \text{ then } \{T(i, j+1)\} \text{ else } \emptyset \\ & \bigcup \{T(i, k), T(k, j+1) : i < k \leq j \text{ and} \\ & \quad \text{Good}_c(i, k) \text{ and } \text{Good}_o(k, j+1)\} \\ & \bigcup \{T(k, j+1) : i < k \leq j \text{ and} \\ & \quad \neg \text{Good}_c(i, k) \text{ and } \text{Good}_o(k, j+1)\} \quad (1) \end{aligned}$$

Equation 1 includes three types of phrases:  $p$  merged with  $w_j$ ,  $p$  split into two parts with the second part merged with  $w_j$ , and finally the case where part of  $p$  is merged with  $w_j$ . Let  $P$  be a phrase assignment over range  $T(I, J)$ . The head,  $\text{head}(P)$ , of  $P$  is the phrase assignment that consists of all phrases in  $P$  except the last one, and the tail,  $\text{tail}(P)$ , of  $P$  is the last phrase of  $P$ .

The non-deterministic phrase assignment extension to word  $w_J$  is given by:

$$P \oplus w_J = \{\text{head}(P) + P_1 : P_1 \in \text{tail}(P) \oplus w_J\}$$

We omit the details of the pseudo code for computing the phrase extension,  $p \oplus w$ , and the phrase assignment extension  $P \oplus w$ , as they are implemented precisely as defined. The only point to note is that  $P \oplus w$  will raise a *Not mergable* exception if  $P \oplus w = \emptyset$ .

## 4.1 An exact solution

We first present an exact solution to Problem 2. Although it correctly computes the stream of annotated phrase assignments over stable ranges, it fails to satisfy the constant time/space complexity requirement. Nonetheless, it serves as the core algorithm, based on which we will construct an online version. The exact solution is shown in Algorithm 1.

PROPOSITION 4.1 (CORRECTNESS). *Algorithm 1 generates frames over all minimally stable ranges.*

PROPOSITION 4.2 (MAXIMAL PHRASE LENGTH). *Let  $\ell(\mathbf{E}) = \max_{e \in \mathbf{E}} \text{length}(\text{text}(e))$ , where  $\text{length}$  is the length in words. Then, for each phrase reported by Algorithm 1, we have:*

$$\text{length}(p) \leq \left\lceil \frac{\ell(\mathbf{E})}{C_{\text{open}}} \right\rceil$$

PROOF PROOF OUTLINE. All  $p$  must be a tail phrase of some  $P$ , thus, by Line 8 of Algorithm 1, it must have an open score  $\geq C_{\text{open}}$ . If  $\text{length}(p) \geq \left\lceil \frac{\ell(\mathbf{E})}{C_{\text{open}}} \right\rceil$ , then it cannot match with any entities and still have the required open score.  $\square$

---

**Algorithm 2** Online annotation

---

```
lines 1...5 of Algorithm 1
if width of  $M > C_{\text{framewidth}}$  then
  yield  $\{\text{head}(P) : P \in M\}$ 
   $M = \{\{\text{tail}(P)\} : P \in M\}$ 
end if
lines 6...13 of Algorithm 1
```

---

## 4.2 Online annotation

Given our emphasis on handling text streams with unbounded length, it is important that the entity annotation algorithm have constant complexity when processing each new word in the stream. Let the *width* of a frame be the number of words of the corresponding minimally stable range.

PROPOSITION 4.3. *With a fixed entity database  $\mathbf{E}$ , computing  $p \oplus w$  requires constant time. The time complexity of Line 3. of Algorithm 1 is  $\mathcal{O}(2^n)$  where  $n$  is the width of  $M$ .*

So, Algorithm 1 is not sufficient to handle streams with long stable ranges. Generally, the width of a minimally stable range is unbounded. Consider the entity database consisting of movies and actors, and the stream ‘jack black jack black jack ...’. Given the actor entity JACK BLACK and the movie titled BLACK JACK, the entire stream has one stable region.

For real-time and interactive entity annotation applications, there should also be a constant bound on the delay between two consecutive frames yielded by the annotation algorithm. Algorithm 1, however, does not have these properties. A frame  $M(I, J)$  is only yielded after the boundary  $J$  of the stable range  $T(I, J)$ . So, the delay would be  $J - I$ .

In order to make Algorithm 1 online and with a lower bound on real-time delay, we construct an approximation algorithm which yields a stream of possibly overlapping frames. By sacrificing the disjointness property of the frames, we can place a bound on the frame width. This is done by modifying Algorithm 1 as shown in Algorithm 2. We refer to the resulting algorithm as the *online annotation* algorithm.

PROPOSITION 4.4. *Each iteration in the for-loop of the online annotation algorithm takes constant time and space.*

PROOF. By induction, it’s easy to see that cardinality of  $M$  is bounded in each iteration, and therefore,  $M'$  can be computed in constant time. The time and space complexity is bounded by  $\max |M| = \Theta(2^{C_{\text{framewidth}}})$ .  $\square$

The bound seems rather excessive. However, our experiments (Section 6, Figure 5(b)) show that,  $\max |M| \ll 2^{C_{\text{framewidth}}}$ . The reason is that, by using  $C_{\text{open}}$  to prune out invalid phrase extensions, and  $C_{\text{closed}}$  to reduce the number of extensions while computing  $p \oplus w$ , we are able to reduce the ambiguity of phrase assignments dramatically from the worst case scenario.

The online annotation algorithm approximates frames over minimally stable regions with frames that *may* overlap. The following result places a bound on the amount of overlap.

PROPOSITION 4.5. *The overlap between the ranges covered by any two adjacent frames yielded by the online annotation algorithm is bounded by a constant  $\left\lceil \frac{\ell(\mathbf{E})}{C_{\text{open}}} \right\rceil$ .*

PROOF OUTLINE. From the construction of Algorithm 2, two adjacent frames overlap by at most one phrase. By Proposition 4.2, the phrase length is at most  $\left\lceil \frac{\ell(\mathbf{E})}{C_{\text{open}}} \right\rceil$ .  $\square$

### 4.3 Similarity, indexing and searching entities

So far, we have treated the similarity measure as a black box. We also omitted details on the retrieval of entities to form the non-deterministic entity annotation of the phrases contributed by the online annotation algorithm.

We choose to utilize the Jaccard similarity of  $q$ -grams of two strings as the string similarity. Let  $\text{Grams}_q(w)$  be the  $q$ -grams of the word  $w$ . The string similarity is given as:

$$\text{SIMILARITY}(w, w') = \frac{\text{Grams}_q(w) \cap \text{Grams}_q(w')}{\text{Grams}_q(w) \cup \text{Grams}_q(w')}$$

Given a choice of  $q$ , entities are indexed as documents. Terms of the document for an entity  $e$  are the  $q$ -grams of  $\text{text}(e)$ , and they are stored in an inverted list text index [9]. The index supports very efficient implementation of a search function, defined as:

$$\begin{aligned} \text{Search} &: \text{Text} \times \mathbb{R} \rightarrow \mathcal{P}_{\text{score}}(\mathbf{E}) \\ &: (w, c) \mapsto \{(e, \text{SIMILARITY}(s, \text{text}(e))) : \\ &\quad e \in \mathbf{E} \text{ and } \text{SIMILARITY}(q, \text{text}(e)) \geq c\} \end{aligned}$$

Recall that the online annotation algorithm computes the scores  $\text{score}(i, j)$  of various phrases  $p = T(i, j)$  when computing the extensions  $P \oplus w$  (Line 3 in Algorithm 1). We memoize the search results during the computation for  $\text{score}(i, j)$ . Let  $H$  be a hash table used to store the memoization. The entry  $H[i, j]$  stores the search result  $\text{Search}(T(i, j), C_{\text{closed}})$ . The score  $\text{score}(i, j)$  is computed as in Algorithm 3. The

---

#### Algorithm 3 $\text{Score}(i, j)$

---

```

1: if  $H[i, j]$  is undefined then
2:    $H[i, j] \leftarrow \text{Search}(T(i, j), C_{\text{closed}})$ 
3: end if
4: return  $\max\{\text{SIMILARITY}(T(i, j), e) : e \in H[i, j]\}$ 

```

---

search operator is relatively expensive because it requires access to the external disk-based index. In order to speed up the search and further utilize existing memoized search results, we introduce a technique to approximate  $\text{score}(i, j)$  by recycling the search results of other relevant phrases in the cache. If  $H[i, j]$  is undefined, instead of committing to a disk access search immediately, we check to see if an approximation to the search result can be constructed from other entries in  $H$ . If by recycling cached search results, we have achieved high enough matching, then the approximation is returned, otherwise a disk-based search is issued. The search approximation algorithm is shown in Algorithm 4.

The computation of the open scores  $\text{score}_o(i, j)$  is done similarly. The only differences are to replace  $C_{\text{closed}}$  with  $C_{\text{open}}$ , and to use the maximal-matching based open similarity measure when comparing strings.

The runtime constants  $C_{\text{closed}}$  and  $C_{\text{open}}$  in the algorithms control the precision and runtime. Figure 1 gives some typical values that are shown to work well in the experiments of Section 6.

## 5. ADAPTIVE ONLINE ANNOTATION

By adaptive online annotation, we mean to design the online annotation algorithm so that it can adapt to changes in

---

#### Algorithm 4 Approximate-search( $T(i, j), C_{\text{closed}}$ )

---

```

1: if  $H[i, j]$  is undefined then
2:   if  $\exists k \in [i, j), H[i, k]$  or  $H[k, j]$  is defined then
3:      $H[i, j] \leftarrow \bigcup_{k \in (i, j)} \{e \in H[i, k] \cup H[k, j] :$ 
4:        $\text{SIMILARITY}(T(i, j), \text{text}(e)) \geq C_{\text{closed}}\}$ 
5:   end if
6:   if  $H[i, j]$  is undefined
7:     or  $(\max_{e \in H[i, j]} \text{SIMILARITY}(T(i, j), e)) \leq C_{\text{apxsearch}}$ 
8:   then
9:      $H[i, j] \leftarrow \text{Search}(T(i, j), C_{\text{closed}})$ 
10:  end if
11: end if
12: return  $H[i, j]$ 

```

---

Parameter	Effect	Discretized values
$C_{\text{closed}}$	Increasing (decreasing) efficiency - reducing (improving) recall	0.5, 0.6, ..., 1.0
$C_{\text{open}}$	Increasing (decreasing) efficiency - reducing (improving) precision and recall	0.3, 0.4, ..., 1.0
$C_{\text{framewidth}}$	Improving efficiency (and accuracy), increasing the memory requirement	10, 15, ..., 30
$C_{\text{approxsearch}}$	Exact is inefficient but accurate, approximate is fast but reduces accuracy.	0.0, 0.1, ..., 1.0
$q$ -gram	Increasing $q$ improves efficiency, $q > 3$ may result in loss of accuracy	3, 4, 5, $\infty$ , (restricted to the available indices).

Figure 1: Summary of runtime parameters

the run-time environment, user preferences and search patterns. Run-time changes include varying word rate of the text stream and the varying processing speed of the CPU. We wish to design the annotation algorithm to self-adjust to these changes. For instance, if the text stream speeds up, the annotation algorithm can still keep up with the annotation by sacrificing in accuracy. Conversely, if the text stream slows down, the annotation algorithm should automatically spend more time in processing to improve the accuracy. User preferences and search patterns allow the online annotation algorithm to personalize the search to individual users. User preference allows certain types of entities to be favored, affecting the rank of the non-deterministic entity annotations of the phrases. Search patterns allow the online annotation algorithm to identify historically popular patterns in the stream, and use them to better predict the non-deterministic phrase assignments.

### 5.1 Runtime adaptation via parameter tuning

There are a number of runtime parameters that affect the time and accuracy of the annotation algorithm. We summarize them in Figure 1. Assume that each parameter is selected from a finite discrete collection. For example, the threshold value  $C_{\text{closed}}$  is to be selected from  $\{0.5, 0.6, \dots, 1.0\}$ . Let  $\{\alpha_k\}$  be the runtime parameters, and  $\mathbf{X}(\alpha_i)$  be the choices of  $\alpha_i$ . A runtime configuration is a tuple  $\vec{v} \in \prod_k \mathbf{X}(\alpha_i)$ . Define the configuration space  $\mathcal{X} = \prod_k \mathbf{X}(\alpha_i)$ . Based on the performance of the online entity annotation algorithm, we wish to dynamically adjust the configuration to (1) minimize the delay of annotations

and (2) maximize the accuracy of the annotations. For each parameter  $\alpha_i$ , can be sorted using an ordering function  $\prec$  from low values that improve accuracy (but the algorithm runs slow) to high values that reduce accuracy (but the algorithm runs fast). For instance,  $C_{\text{closed}} = 0.5$  allows approximate string matching, which increases the size of  $p \oplus w$  thus making the annotation algorithm slower. On the other hand,  $C_{\text{closed}} = 1.0$  uses exact string matching, reducing  $p \oplus w$ , possibly missing relevant entities, but improves the time complexity of the algorithm. Thus, it is desirable to select the *small* values w.r.t.  $\prec$  from a set of possible settings  $X_i$  while ensuring a minimal performance of the annotation given by the word rate of the text stream. If one is able to observe the resulting accuracy, a multi-variable closed loop controller can be designed using feedback control. However, this is not possible for our scenario, as we cannot realistically expect the user to provide query feedback for all non-deterministically annotated frames. We propose an open-loop best-effort controller.

First, we group the parameters into two groups:

$$\begin{aligned} \mathbf{C}_1 &= \{C_{\text{open}}, C_{\text{closed}}, C_{\text{framewidth}}, C_{\text{approxsearch}}\} \\ \mathbf{C}_2 &= \{q\text{-gram}\} \end{aligned}$$

The first group  $\mathbf{C}_1$  are parameters that are easily updatable, incurring no overhead or external behavioural changes. In contrast, the choice of  $q$  requires the search function to use a different text index, so there is significant overhead associated with updates to the value of  $q$  at runtime.

We define **Zorder**( $X_1, X_2, \dots, X_k$ ) to be the z-ordering [11] of the ordered sets  $X_i$ . Z-ordering in a multidimensional grid has the desirable property that it will equally enumerate over values in each  $X_i$  from small to large.

We define **Lexico**( $X_1, X_2, \dots, X_k$ ) to be the lexicographical ordering of the ordered sets  $X_i$ . **Lexico**( $X_1, X_2$ ) will exhaust all possibilities in  $X_1$  before changing the values in  $X_2$ , thus, at runtime, the adaption of parameters in the order of **Lexico**( $X_1, X_2$ ) has the desirable property that  $X_2$  remains fixed for as long as possible. Using **Zorder** and **Lexico**, we linearly order the entire multidimensional parameter space into a single dimension:

$$(\mathcal{X}, \prec) = \text{Lexico}(\text{Zorder}(\mathbf{C}_1), \mathbf{C}_2)$$

The adaptive parameter selection algorithm uses the linear ordering to select the next runtime parameter. It detects the current text word rate, and the annotation rate, and updates the current runtime parameter by either choosing smaller values or larger values in  $(\mathcal{X}, \prec)$ .

$$\begin{aligned} r_{\text{text}} &= \text{current word rate of the text stream} \\ r_{\text{entity}} &= \text{word rate in the past } N \text{ frames} \\ \vec{v} &= \text{values of the current runtime parameter} \\ \vec{v}' &= \begin{cases} \text{next in } (\mathcal{X}, \prec) & \text{if } r_{\text{text}} - r_{\text{entity}} > \Delta r \\ \text{previous in } (\mathcal{X}, \prec) & \text{if } r_{\text{entity}} - r_{\text{text}} > \Delta r \\ \vec{v} & \text{otherwise} \end{cases} \end{aligned}$$

where  $N$  is the window size used to estimate the word rate of the annotation algorithm, and  $\Delta r$  is the threshold for changing the the run-time parameter.

## 5.2 User preference

Generally, the database of entities span over multiple types. For example, in the IMDB data set, we have entity types *MOVIE*, *ACTOR*, *LOCATION*, *TRIVIA*, etc. Users have different preferences over the types of entities. A typical user may only be interested in the movie and actor entities, while a movie buff may be interested in the trivia and location entities as well. Users may also exhibit certain search patterns out of habit. For instance, one user may have the habit of mentioning a list of actors and then movies, while another user may list movies and actors in random order. In this section, we present the modeling of user preferences and search patterns, training methods and how they are incorporated into the online entity search algorithm. For feedback training, we assume the user will select the *correct* phrase assignment  $P_M^*$  over each region  $M$  in the stream. Furthermore, the user will also indicate the *correct* entity annotation for each phrase in  $P^*$ . Given that the search algorithm yields a stream of regions, we choose to concatenate  $P^* = P_{M_1}^* + P_{M_2}^* + \dots$ . Since each phrase  $p \in P^*$  is assumed to have been disambiguated into a unique entity  $e_p^*$ , we assume that the user feedback is a list of entities  $E^*$ . The user preference is modeled as a simple preference function:

$$\text{pref} : \text{Types}(\mathbf{E}) \rightarrow \mathbb{R}^+$$

**Training:** Given that we have the list of entities  $E^*$ , it is easy to train for a preference function. We apply Laplace’s law [10] for unobserved entity types:

$$\text{pref}(t) = \frac{|\{e \in E^* : \text{type}(e) = t\}| + 1}{|E^*| + |\text{Types}(\mathbf{E})|}$$

**Preference based search:** The similarity measure between closed phrases and an entity is modified to be:

$$\begin{aligned} &\text{SIMILARITY}(T(i, j), e | \text{pref}) \\ &= \text{SIMILARITY}(T(i, j), \text{text}(e)) \cdot \text{pref}(\text{type}(e)) \end{aligned}$$

The search algorithm simply utilizes the preference specific similarity function.

## 6. IMPLEMENTATION AND EXPERIMENTAL EVALUATIONS

We have conducted extensive experimental evaluation on the implementation of our online entity annotation algorithm against a variety of data sets. The algorithm is implemented in Python. We used the open source search engine, Xapian<sup>7</sup>, to implement the search function in Section 4.3. We modified Xapian’s ranking function to rank the search results according to the Jaccard similarity measure. In order to also compute the *open* similarity measure, we perform ad-hoc re-ranking, when necessary, inside the Python program.

**The data sets.** We have collected three data sets to evaluate the algorithm. They are listed in Figure 3. IMDB consists of entity types: movies, actors, actresses and production companies. The *Companies* data set consists of nine different entity types ranging from companies, products, employees, boards of directors, etc. The final data set is an undergraduate teaching schedule consisting of course code, course title, description, room number, and instructor names, etc.

**Experimental design.** From the data sets, we generate a text stream consisting of entity names (altered with

<sup>7</sup>The Xapian Project: <http://xapian.org>

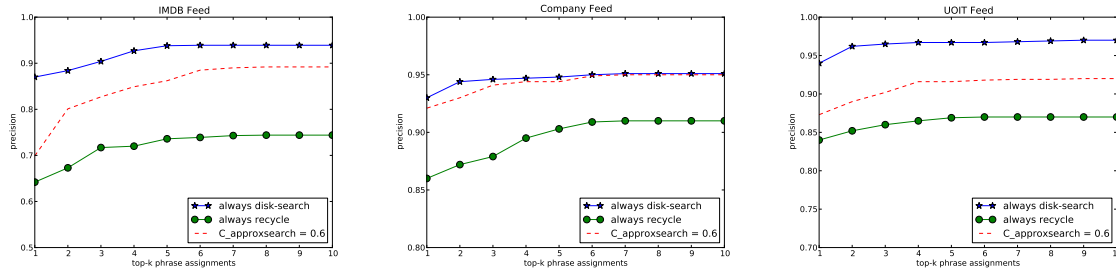


Figure 2: Precision with respect to top- $k$  phrase assignments

Data set	# of entities	# of types
IMDB	2,300 K	4
Companies	77 K	9
University course info.	6 K	8

Figure 3: Data sets used

spelling errors and truncation). Entities are randomly sampled from the data sets, and their text values are altered by permutation of randomly selected letters by substitution or deletion. The probability of permutation is referred to as the *spelling error*. This is to simulate the spelling errors for humanly generated text streams. The resulting text stream is processed by the online entity annotation algorithm. Based on the annotated frames generated by the algorithm, we measure the precision and recall of the annotation, and the performance of the annotation. The runtime parameters are varied to evaluate the effect of the parameter tuning on the performance of the algorithm. We have also experimented with the user preference learning (Section 5.2) and its effect on precision. Unless otherwise specified, the default values used for the runtime parameters are:  $C_{\text{closed}} = 0.6$ ,  $C_{\text{open}} = 0.4$ ,  $C_{\text{framewidth}} = 20$ .

**Precision and recall.** Recall that the annotation algorithm yields a stream of frames. Each frame consists of a ranked list of phrase assignments. The  $k$ -correctness of each frame, as a function of an integer  $k > 0$ , is defined as follows: if *any of the top- $k$  phrase assignment* matches with the sampled entities in the same stream range, then the frame is *correct*; otherwise, the frame is *incorrect*. The  $k$ -precision is the percentage of correct matches. Similarly, the  $k$ -recall is the percentage of sampled entities that are found in the top- $k$  phrase assignment of the corresponding frame.

**Experimental results.** The precision for different  $k$  values for all data sets are shown in Figure 2. The recall values are quite similar, so we omitted the plots. In Figure 2, we have also included the  $k$ -precision of always using disk-based index search, always recycling previously cached search result, and the hybrid of  $C_{\text{approxsearch}} = 0.6$ . One can see that the hybrid search (with recycling previous results) yields comparable accuracy comparing to the more costly naive method.

Since, we are dealing with approximate string matching, our annotation algorithm can handle spelling error introduced by human errors. The precision and recall for different spelling errors of the IMDB based text stream is shown in Figure 4(a).

The histogram performance of the annotation algorithm for the IMDB data set is shown if Figure 5(a). The x-axis is

the time it takes to process each additional word (seconds), and the y-axis is the normalized count over 1000 words. One can see that approximate search by recycling dramatically improves the rate of annotation.

We experimented with the effect of runtime parameters on the precision, recall and annotation rate of the algorithm. Figure 4(b) shows the effect of  $C_{\text{closed}}$  on the precision and recall (with  $k = 1$ ). Note, by increasing  $C_{\text{closed}}$  close to 1.0, it greatly improves the precision because it requires stronger string similarity, and thus recall is greater deduced.

Figure 5(b) shows the distribution of the number of phrase assignments in frames (frame cardinality) for the IMDB stream with the  $C_{\text{framewidth}} = 50$ . (Distributions for University and Company data sets are similar, hence omitted.) Recall from Section 4, the cardinality is loosely bounded by  $2^{C_{\text{framewidth}}}$ . The actual observations show that, in practice, the cardinality is quite small.

In order to study the effect of parameter tuning to trade precision for speed, we have sampled the parameter space  $\mathcal{X} = \prod_k (\mathbf{X}(\alpha_i))$ , where  $\mathbf{X}(\alpha_i)$  is the discretized parameter values of parameter  $\alpha_i$  (Section 5.1). For each random sampled setting, we measured the resulting precision and annotation rate (words/second). The result is plotted in Figure 5(c). The most effective parameter to improve annotation rate is  $C_{\text{approxsearch}}$ , showing that recycling search results is an important method to boost the search speed.

Finally, we experimented with learning of user preferences. For the IMDB database, we restricted the samples of entities to only *actor names*, thus excluding *movies* and *actress names*. Without learning the user preference, the search function returns results containing all entity types, thus reducing the precision. By learning the user preference, the similarity measures of the entities that are *not* actor names are greatly suppressed, and are eliminated by the threshold  $C_{\text{closed}}$ . Figure 4(c) shows that, with learning, we can improve precision greatly.

## 7. CONCLUSION AND FUTURE WORK

We have presented an online entity annotation algorithm which correlates an unbounded text stream with databases of entities by means of non-deterministic annotation. Our algorithm produces *frames* which non-deterministically identify interesting *phrases* in the stream along with relevant entities in a ranked order. Our work distinguishes itself from existing literature by performing entity annotation in a completely online fashion. The algorithm is capable of adapting to varying text word rate by trading off precision with per-

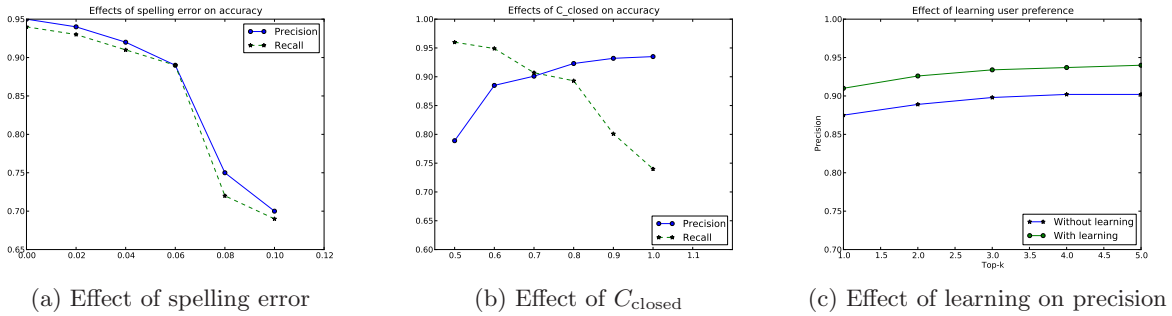


Figure 4: Effects on accuracy

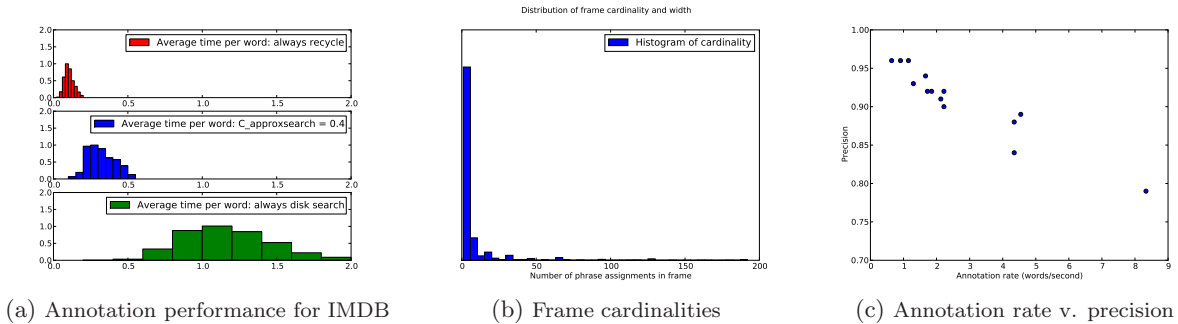


Figure 5: Runtime evaluation of the annotation

formance. The algorithm can also learn the user preference if query feedback is available.

Currently, we do not consider relations among entities. However, in many applications, entity relations are also an important factor in deciding the relevance of an entity. For future work, we would like to extend the scoring function and the online algorithm to incorporate known relations of entities.

## 8. REFERENCES

- [1] S. Agrawal, K. Chakrabarti, S. Chaudhuri, and V. Ganti. Scalable Ad-Hoc Entity Extraction From Text Collections. *PVLDB*, 1(1):945–957, 2008.
- [2] A. Chandel, P. C. Nagesh, and S. Sarawagi. Efficient Batch Top-k Search for Dictionary-based Entity Recognition. In *ICDE*, page 28, 2006.
- [3] S. Chaudhuri, V. Ganti, and D. Xin. Exploiting Web Search To Generate Synonyms For Entities. In *WWW*, pages 151–160, 2009.
- [4] W. W. Cohen and S. Sarawagi. Exploiting Dictionaries in Named Entity Extraction: Combining Semi-Markov Extraction Processes and Data Integration Methods. In *KDD*, pages 89–98, 2004.
- [5] O. Etzioni, M. J. Cafarella, D. Downey, A. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates. Unsupervised Named-Entity Extraction from the Web: An Experimental Study. *Artif. Intell.*, 165(1):91–134, 2005.
- [6] S. Ji, G. Li, C. Li, and J. Feng. Efficient Interactive Fuzzy Keyword Search. In *WWW*, pages 371–380, 2009.
- [7] William Kocay and Donald Kreher. *Graphs, algorithms, and optimization*. Chapman and Hall, 2005.
- [8] F. Liu, C. Yu, W. Meng, and A. Chowdhury. Effective Keyword Search In Relational Databases. In *SIGMOD*, pages 563–574, 2006.
- [9] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge Univ. Press, 2008.
- [10] C. D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. The MIT Press, 1999.
- [11] J. A. Orenstein and T. H. Merrett. A class of data structures for associative searching. In *PODS*, pages 181–190, 1984.
- [12] K. Q. Pu and X. Yu. Keyword Query Cleaning. *PVLDB*, 1(1):909–920, 2008.
- [13] V. S. Uren, P. Cimiano, J. Iria, S. Handschuh, M. Vargas-Vera, E. Motta, and F. Ciravegna. Semantic Annotation for Knowledge Management: Requirements and a Survey of the State of the Art. *JWS*, 4(1):14–28, 2006.
- [14] W. Wang, C. Xiao, X. Lin, and C. Zhang. Efficient Approximate Entity Extraction with Edit Distance Constraints. In *SIGMOD*, pages 759–770, 2009.
- [15] Y. Yang, N. Bansal, W. Dakka, P. G. Ipeirotis, N. Koudas, and D. Papadias. Query by Document. In *WSDM*, pages 34–43, 2009.
- [16] J. X. Yu, L. Qin, and L. Chang. Keyword Search in Databases. *Synthesis Lectures on Data Management*, 1(1):1–155, 2009.