2ED: An Efficient Entity Extraction Algorithm using Two-Level Edit-Distance

Zeyi Wen[†1], Dong Deng[‡2], Rui Zhang[§3], Kotagiri Ramamohanarao[§4]

[†]*National University of Singapore,* [§]*The University of Melbourne*

[1]wenzy@comp.nus.edu.sg, {[3]rui.zhang, [4]kotagiri}@unimelb.edu.au

[‡]*Rutgers University & Inception Institute of Artificial Intelligence*

[2]buaasoftdavid@gmail.com

***Abstract*—Entity extraction is fundamental to many text mining tasks such as organisation name recognition. A popular approach to entity extraction is based on string matching against a dictionary of known entities. For approximate entity extraction from free text, considering solely character-based or solely token-based similarity cannot simultaneously deal with minor name variations at token-level and typos at character-level. Moreover, the tolerance of mismatch in character-level may be different from that in token-level, and the tolerance thresholds of the two levels should be able to be customised individually. In this paper, we propose an efficient character-level and token-level edit-distance based algorithm called FuzzyED. To improve the efficiency of FuzzyED, we develop various novel techniques including (i) a spanning-based candidate sub-string producing technique, (ii) a lower bound dissimilarity to determine the boundaries of candidate sub-strings, (iii) a core token based technique that makes use of the importance of tokens to reduce the number of unpromising candidate sub-strings, and (iv) a shrinking technique to reuse computation. Empirical results on real world datasets show that FuzzyED can efficiently extract entities and produce a high F_1 score in the range of [0.91, 0.97].**

I. Introduction

In data integration and text mining, a primitive task is entity extraction—the recognition of the names of entities such as people, locations and organisations—in a free text document [1], [2]. A common approach to entity extraction is to compare sub-strings of a document (hereafter "*candidate sub-strings*" or simply "*candidates*") against a dictionary of entities [3]. This approach needs to handle the following two issues. (i) Typos may appear in documents, e.g. "Melbourne" written as "Melbounre". (ii) Different names may refer to the same entity, e.g. "Melbounre University" is the same as "The University of Melbourne". It is challenging to address the two issues in the context of free text, due to the difficulty in deciding the boundaries of a candidate sub-string. As a result, the number of candidate sub-strings is very large, and all those candidate sub-strings need to match against the dictionary.

Previous methods [4], [5], [6] using only character-based or token-based (i.e. one-level) similarity cannot handle *both* of the issues in the free text context. Moreover, the tolerance of mismatch in character-level is usually different from that in token-level depending on applications. Hence, the tolerance thresholds of the two levels should be customised individually by the domain experts. To be more concrete, when analysing BBC news articles, data mining practitioners may want the entity extraction algorithm to tolerate less in character-level and tolerate more in token-level. In comparison, when modelling online product reviews, data mining practitioners may want to be more tolerant in both character-level and token-level. Using only character-based or token-based similarity cannot support the customisation on the two tolerance thresholds.

To extract approximate entities efficiently and effectively, we propose an algorithm based on character-level and token-level edit-distance (i.e. two-level edit-distance) which we call *FuzzyED*. Our key idea is to measure both the character and token similarity via edit-distance. As a result, FuzzyED natively supports the customisation of the tolerance thresholds of mismatch on character-level and token-level individually. However, a naive implementation of two-level edit-distance for entity extraction is inefficient due to the huge computation cost and numerous candidate sub-strings for evaluation. For achieving a fast entity extraction algorithm, we develop various novel techniques to improve the efficiency of FuzzyED. To summarise, we make the following two major contributions in this paper. First, we propose an efficient algorithm based on two-level edit-distance for approximate entity extraction. Second, to reduce the number of unpromising candidate sub-strings in the similarity evaluation, we propose various novel techniques including (i) a spanning-based candidate sub-string producing technique, (ii) a lower bound dissimilarity to determine the boundaries of candidate sub-strings, (iii) a core token based technique that makes use of the importance of tokens to further reduce the number of unpromising candidate sub-strings, and (iv) a shrinking technique to reuse computation during candidate sub-string producing.

We conduct experiments to validate the efficiency and effectiveness of FuzzyED. Our experimental results show that FuzzyED is fast and achieves a high F_1 score in the range of [0.91, 0.97]. The remainder of this paper is structured as follows. We first present the key preliminaries in Section II. Then, we discuss the related work on approximate entity extraction in Section III and elaborate the techniques of FuzzyED in detail in Section IV. Our comprehensive experimental study is provided in Section V. Finally, we conclude the paper in Section VI.

TABLE I
FREQUENTLY USED SYMBOLS

t, e, s	a token, an entity token and a text token
$idf(t)$, $w(t)$	IDF and the weight of t, respectively
$\mathcal{E}$, $\mathcal{S}$, $\mathcal{E}_i$, $\mathcal{S}_j$	an entity, sub-string candidate, the i^{th} token of $\mathcal{E}$, and the j^{th} token of $\mathcal{S}$, respectively
$eds(e, s)$	the edit similarity of e and s
τ, δ	token and entity edit similarity thresholds
$\mathcal{C}$	a set of core tokens of $\mathcal{E}$
$\mathcal{C}_i$	the i^{th} core token in $\mathcal{C}$

II. PRELIMINARIES

For ease of presentation, a token (e.g. word) of a candidate sub-string is called a *text token*. Similarly, we call a token of an entity in the dictionary an *entity token*. Some frequently used symbols in the rest of the paper are summarised in Table I. In this section, we provide background knowledge of edit-distance, edit similarity, and techniques of matching text tokens against entities.

A. Edit-distance and edit similarity

Without loss of generality, we assume that all the edit operations have the same cost in this paper. Formally, given two tokens e and s, the edit similarity $eds(e, s)$ can be computed as follows.

$$eds(e, s) = 1 - \frac{ed(e, s)}{\max\{|e|, |s|\}} \tag{1}$$

where $ed(e, s)$ is the edit-distance between the two tokens; $|e|$ and $|s|$ are the number of characters in e and s, respectively. The computation of edit-distance is a well studied problem, and a common approach to compute edit-distance of two tokens is through dynamic programming [7].

Note that edit-distance and edit similarity can be applied to token-level. We postpone our definition of the cost of token-level edit operations and edit similarity until Section IV.

B. Assigning weights to tokens

In many applications, the tokens in an entity (or a candidate sub-string) have different importance, also called *weights*, in the entity (or the candidate sub-string). In this paper, we focus our presentation on using IDF [8] to measure the weights of tokens, although other ways of measuring the weights of tokens (e.g. TF-PDF [9]) can be straightforwardly integrated into our proposed algorithm. In the approximate entity extraction problem, the dictionary is known a priori and documents are unknown beforehand. Hence, we compute the IDF value of a token based on the dictionary. Specifically, given the dictionary with N entities and a token t, we count the number N_t of entities that contain t to serve as the "document frequency" of the token. Then, the IDF value of t is computed by $idf(t) = \log \frac{N}{N_t + 1}$. The total IDF value of a set of tokens $\mathcal{A}$ is the sum of the IDF values of all the tokens in $\mathcal{A}$, and can be computed as follows.

$$T_{idf}(\mathcal{A}) = \sum_{t \in \mathcal{A}} idf(t) \tag{2}$$

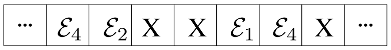

Fig. 1. All the matched text tokens to $\mathcal{E}$

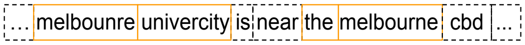

Fig. 2. All the matched text tokens to *the university of melbourne*

After computing $idf(t)$ and $T_{idf}(\mathcal{A})$, we can compute the weight of the token t by the equation below.

$$w(t) = \frac{idf(t)}{T_{idf}(\mathcal{A})} \tag{3}$$

Note that $\mathcal{A}$ can be either the entity $\mathcal{E}$ or the sub-string candidate $\mathcal{S}$. We define the total weight of a subset $\mathcal{A}'$ of tokens in $\mathcal{A}$ (i.e. $\mathcal{A}' \subseteq \mathcal{A}$) as follows.

$$T_w(\mathcal{A}') = \sum_{t \in \mathcal{A}'} w(t) \tag{4}$$

C. Matching text tokens against entities

Since we are interested in extracting entities from documents (i.e. free text), the first step is to find in the documents the tokens that approximately match to the tokens of an entity in the dictionary. There are many efficient algorithms that can check if two tokens are matched approximately [10]. In this paper, we use Li et al.'s algorithm [11] for finding all the matched tokens in a document to an entity. The technical details about how Li et al.'s algorithm works are unimportant for understanding our algorithms, and hence are omitted. Here, we briefly explain the results produced by the algorithm. Figure 1 gives example results of the matched tokens in a document. In the example, the dictionary contains an entity $\mathcal{E}$ which contains four tokens $\mathcal{E}_1$, $\mathcal{E}_2$, $\mathcal{E}_3$ and $\mathcal{E}_4$. The document is represented by a row, and each cell of the row contains the matching information to the entity. A cell containing "X" indicates that the position[1] does not match any token of the entity; a cell containing $\mathcal{E}_i$ indicates that the text token at this position matches the i^{th} token of the entity $\mathcal{E}$. For ease of presentation, we call all the positions marked with $\mathcal{E}_i$ shown in Figure 1 a **position list**.

Example 1: Given a document $\mathcal{D}$ = "... *melbounre univercity is near the melbourne cbd* ..." and an entity $\mathcal{E}$ = "the university of melbourne", then each token of $\mathcal{E}$ is $\mathcal{E}_1$ ="the", $\mathcal{E}_2$ = "university", $\mathcal{E}_3$ = "of", and $\mathcal{E}_4$ ="melbourne" (cf. Figure 2). After identifying all the matched token of $\mathcal{E}$ in $\mathcal{D}$, we can formally represent $\mathcal{D}$ as "... $\mathcal{E}_4$ $\mathcal{E}_2$ X X $\mathcal{E}_1$ $\mathcal{E}_4$ X ...".

III. RELATED WORK

The approximate entity extraction problem can be modelled as a string matching problem [10]. Here, we focus on the relatively recent related work.

[1] For ease of presentation, we refer "the position" to "the token at the position of the document".

A. Related work on entity extraction

Gattani et al. developed a dictionary-based algorithm for entity extraction with exact matching [12]. Kim et al. [13] proposed a memory efficient indexing approach for string matching using character-based similarity. Wang et al. proposed an approximate entity extraction algorithm using neighbourhood generation [4]. Deng et al. [5] designed an efficient algorithm for approximate entity extraction based on trie tree index. Kim and Shim proposed an algorithm that finds from a document top-k most similar candidate sub-strings to an entity [14]. A more recent study [15] presents techniques to find duplicated sub-strings between two documents using token-level similarity. All these algorithms use one-level, i.e. either character-based or token-based, similarity to find similar entities (or candidate sub-strings) in documents.

Some existing studies [16], [17], [18] designed similarity functions and indexing techniques for the string similarity search problem. Cohen et al. [19] developed an open source software toolkit, which supports different similarity functions, for measuring the similarity between two strings. Pappu et al. [20] designed a lightweight system for multilingual entity extraction. Chakrabariti et al. [6] proposed a filter using the token-based similarity to classify candidate sub-strings into two classes: valid candidate sub-strings that may match some entities in the dictionary; invalid candidate sub-strings that do not match any entities in the dictionary. We use Chakrabariti et al.'s filter in our proposed algorithm discussed in Section IV and the baseline discussed in Section V for fair comparison.

Carreras et al. proposed an Adaboost based approach for named entity extraction [21]. Their key idea is to extract entities using two classifiers: a local classifier for detecting if a token belongs to a named entity; a global classifier for detecting if a candidate sub-string is a named entity. Jain and Pennacchiotti [22] proposed an approach using heuristics (e.g. tokens with first letter capitalised) to extract entities from query log, and then the extracted entities are grouped into different clusters and assigned labels accordingly. Cohen and Sarawagi [23] designed an algorithm using the Markov model for entity extraction. The algorithm has two main phases. First, a label (e.g. person name) is assigned to each token based on dictionaries/heuristics. Then, the Markov model is trained and used to predict the entity probability for each candidate sub-string based on the token labels. One major limitation of the abovementioned approaches is that they require significant amount of human effort to collect training datasets and/or to tune heuristics. When the training datasets are not available, it is impossible to extract entities using machine learning. Our work is to address entity extraction problems without the training datasets.

B. Related work on string similarity joins

The string similarity join problems are well studied in the field of text mining and data cleaning [24], [25], [26]. A string similarity join finds similar pairs between two collections of strings. The assumption is that the candidate sub-strings are already available. This assumption is invalid in the entity extraction problems where the candidate sub-strings need to be obtained from free text. Nevertheless, our baseline is inspired by the Fuzzy Jaccard algorithm for solving string similarity join problems. In the following, we present the key related work in string similarity joins.

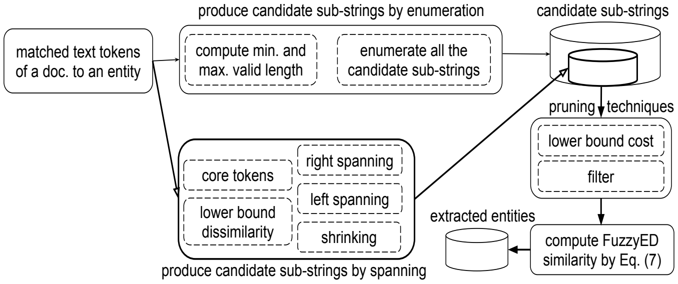
produce candidate sub-strings by enumeration
candidate sub-strings
matched text tokens of a doc. to an entity
compute min. and max. valid length
enumerate all the candidate sub-strings
pruning techniques
lower bound cost
filter
core tokens
right spanning
left spanning
lower bound dissimilarity
shrinking
produce candidate sub-strings by spanning
extracted entities
compute FuzzyED similarity by Eq. (7)

Fig. 3. Overview of the process of FuzzyED

Many studies [27], [28] have been dedicated to solving the string similarity join problems more efficiently and effectively. However, those studies exploit only the character-based similarity for the string similarity join problems. A more recent work [29] studies approximate string joins with abbreviation. Fuzzy Jaccard [30], [31] uses both character-based and token-based similarity for the string similarity join problems. It can be extended to extract entities when it equips with our proposed candidate sub-string producing techniques discussed in the next section. We will discuss the extension of Fuzzy Jaccard for entity extraction in Section V when we present our baseline.

IV. Our FuzzyED algorithm

In this section, we elaborate the details of our proposed algorithm for approximate entity extraction based on two-level (i.e. character-level and token-level) edit-distance, and we call it Fuzzy Edit-Distance (FuzzyED for short). FuzzyED natively allows domain experts to customise the tolerance thresholds of the two levels individually.

A. Overview of FuzzyED

The overview of the FuzzyED algorithm is shown in Figure 3. To begin with, the matched text tokens of a document to an entity is obtained. In this paper, we use Li et al.'s algorithm [11] to obtain the matched text tokens as we have discussed in Section II-C, although other algorithms of finding matched tokens between a document and an entity work just fine with FuzzyED.

In the second step, we need to find out all the candidate sub-strings which are potentially recognised as "entity". In order to do that, we first propose a method to produce candidate sub-strings by enumeration, which exploits the minimum/maximum valid matching length to reduce the number of candidate sub-strings. However, due to the high computation cost of two-level edit-distance and numerous candidate sub-strings for evaluation, producing candidate sub-strings by enumeration is inefficient. To address the inefficiency problem, we propose a method which is equipped with various novel techniques. Firstly, we develop a core token based technique that makes use of the importance of tokens to reduce the number of unpromising candidate sub-strings. Secondly, we explore the lower bound dissimilarity of two-level edit-distance to determine the boundaries of candidate sub-strings. Thirdly, we exploit right spanning and left spanning to produce candidate sub-strings, which avoids producing many unpromising candidate sub-strings. Finally, we propose a shrinking technique to reuse the computation during the candidate sub-string producing with spanning.

After the candidate sub-strings are produced, we exploit techniques to further prune the candidate sub-strings that will not be recognised as "entity". The pruning techniques include a lower bound cost of edit operations and a filter (discussed later in Section IV-E). Then, we measure the FuzzyED similarity for the remaining candidate sub-strings, and those sub-strings with similarity scores above a threshold are recognised as "entity".

Next, we present the details of the whole FuzzyED algorithm. We first define the cost of token-level edit operations and the similarity of FuzzyED. We then present two techniques for producing candidate sub-strings, one based on enumeration and one based on spanning. Finally, we describe the pruning techniques for further improvement.

B. Token-level edit operations and FuzzyED similarity

Here, we define the cost of the token-level edit operations including deletion, insertion and substitution. The key idea is similar to the traditional edit-distance. After defining the token-level edit operations, we define the similarity function used in FuzzyED. The similarity score is bounded in the domain of [0,1] by forcing the negative scores back to 0 and large scores back to 1.

1) Cost of token-level edit operations: FuzzyED needs to perform two types of edit-distance: the character-based edit-distance and the token-based edit-distance. As we have discussed the character-level edit-distance in Section II-A, here we provide details of the token-level edit-distance.

We formulate the total cost of transforming a candidate sub-string $\mathcal{S}$ to an entity $\mathcal{E}$ using the following equation.

$$FuzzyED(\mathcal{E},\mathcal{S}) = C_D(\mathcal{S}) + C_I(\mathcal{E}) + C_S(\mathcal{E},\mathcal{S}) \qquad (5)$$

where $C_D(\mathcal{S})$ is the total deletion cost of removing text tokens from $\mathcal{S}$; $C_I(\mathcal{E})$ is the total insertion cost of inserting entity tokens of $\mathcal{E}$ to $\mathcal{S}$; $C_S(\mathcal{E},\mathcal{S})$ is the total substitution cost of $\mathcal{E}$ and $\mathcal{S}$. We let $\mathcal{S}'$ be a subset of tokens in $\mathcal{S}$ that match to $\mathcal{E}$; $\mathcal{E}'$ denotes tokens that are matched by $\mathcal{S}'$.

Deletion: The total deletion cost is computed by $C_D(\mathcal{S}) = T_w(\mathcal{S} \setminus \mathcal{S}')$, where $\mathcal{S} \setminus \mathcal{S}'$ is a subset of the tokens in $\mathcal{S}$ (i.e. $\mathcal{S} \setminus \mathcal{S}' \subseteq \mathcal{S}$) that need to be deleted from $\mathcal{S}$.

Insertion: The total insertion cost is computed by $C_I(\mathcal{E}) = T_w(\mathcal{E} \setminus \mathcal{E}')$, where $\mathcal{E} \setminus \mathcal{E}'$ is a subset of tokens in $\mathcal{E}$ (i.e. $\mathcal{E} \setminus \mathcal{E}' \subseteq \mathcal{E}$) that needs to be inserted to $\mathcal{S}$.

Substitution: The total substitution cost is computed by:

$$C_S(\mathcal{E},\mathcal{S}) = \sum_{e \in \mathcal{E}', s \in \mathcal{S}'} (1 - eds(e,s)) \times (w(e) + w(s)) \qquad (6)$$

where s is a text token ($s \in \mathcal{S}'$) that matches entity token e.

Following the common practice of computing edit-distance, we propose to use a dynamic programming based algorithm [10] to compute Equation (5) to measure the cost of the longest sub-string of $\mathcal{S}$ that is the most similar to the entity $\mathcal{E}$. The time complexity of the algorithm is $\mathcal{O}(mn)$, where m and n are the number of tokens of $\mathcal{E}$ and $\mathcal{S}$, respectively.

2) Computing the FuzzyED similarity: After computing the total edit cost in Equation (5), we can compute the FuzzyED similarity using the following equation.

$$FuzzyEDS(\mathcal{E},\mathcal{S}) = \begin{cases} 0 & \text{if } FuzzyED(\mathcal{E},\mathcal{S}) \geq 1, \\ 1 - FuzzyED(\mathcal{E},\mathcal{S}) & \text{otherwise.} \end{cases} \qquad (7)$$

Note that the substitution cost may be larger than 1 when the token edit similarity threshold τ is less than 0.5 (cf. Equation (6)) which results in $FuzzyEDS(\mathcal{E},\mathcal{S}) > 1$.

C. Producing candidate sub-strings by enumeration

We present the first approach to produce candidate sub-strings for evaluation here. The key idea of this approach is to enumerate all the possible sub-strings of a document. This enumeration results in a large number of candidate sub-strings, many of which are not matched to the entity. To filter out those sub-strings, we deduce two propositions of valid matching length. The intuition of the two propositions is that very short or very long candidate sub-strings will not match the entity. We explain the details of this enumeration based approach and the two propositions in the following.

Recall that a position list corresponds to the matches between the document and an entity (cf. Section II-C). Given a position list, we can obtain the candidate sub-strings by enumeration, i.e. all candidate sub-strings with one matched token, two matched tokens, three matched tokens, etc. This enumeration produces $\frac{k(1+k)}{2}$ candidate sub-strings in total, where k is the length of the position list (i.e. the number of tokens that match the entity in the document). Among the $\frac{k(1+k)}{2}$ candidate sub-strings, many of them tend to be unpromising candidate sub-strings. For example, a candidate sub-string with only one matched token is unlikely to match an entity of ten tokens with threshold $\delta = 0.8$. To produce fewer unpromising candidate sub-strings, we propose a lower bound and upper bound (denoted by l and u, respectively) for the number of matched tokens. We refer the candidate sub-strings with length in the domain $[l,u]$ to candidate sub-strings of *valid matching length.* The intuition of the valid matching length is that very short or very long candidate sub-strings will not match the entity with threshold of δ. Next, we present two propositions for the minimum and maximum valid matching length (i.e. the lower and upper bound).

1) Two propositions of the valid matching length: For ease of presentation, we classify the text tokens of a sub-string candidate $\mathcal{S}$ into three types. (1) *Unmatched text tokens* denoted by $\hat{\mathcal{S}}$. (2) *Redundant matched text tokens* denoted by $\mathcal{S}''$: the text tokens match the entity tokens but are finally removed (by the maximum weight matching algorithm). (3)

Valid matched text tokens denoted by $\mathcal{S}'$: the text tokens match the entity tokens and are not redundant. Please note that only the redundant matched text tokens and valid matched text tokens are in the position list.

The minimum valid matching length l: Suppose a sub-string candidate $\mathcal{S}$ has only l tokens that match the entity $\mathcal{E}$, i.e. the similarity of $\mathcal{S}$ and $\mathcal{E}$ is not smaller than δ. If l is the minimum valid matching length, the following proposition must be true:

Proposition 1. *All the l text tokens are (i) exactly matched to a token of the entity and (ii) valid matched text tokens.*

The proof is straightforward and hence omitted. According to the proposition, we have $\mathcal{S} = \mathcal{S}'$ and $T_w(\mathcal{S}) = T_w(\mathcal{S}') = 1$ (cf. Equations (4)), where $\mathcal{S}'$ denotes the valid matched tokens in $\mathcal{S}$.

The maximum valid matching length u: Suppose a sub-string candidate $\mathcal{S}$ has u tokens matched the entity with similarity score not smaller than δ. If u is the maximum valid matching length, the following proposition must be true:

Proposition 2. *All the tokens of the entity are exactly matched.*

The proof is straightforward and hence omitted. From the above proposition, we have $T_w(\mathcal{E}) = T_w(\mathcal{E}') = 1$ (cf. Equations (4)), where $\mathcal{E}'$ denotes all the matched tokens in $\mathcal{E}$. Next, we compute a domain $[l, u]$ for the valid matching length for FuzzyED.

The minimum valid matching length l: According to Proposition 1, the substitution and deletion cost are zero, and only the insertion cost is involved in transforming $\mathcal{S}$ to $\mathcal{E}$. Therefore, the total cost $FuzzyED(\mathcal{E}, \mathcal{S})$ equals to the insertion cost $T_w(\mathcal{E} \setminus \mathcal{E}')$ where $\mathcal{E} \setminus \mathcal{E}'$ is a subset of the tokens in $\mathcal{E}$ needed to be inserted to $\mathcal{S}$. According to Equation (7), the similarity score is $1 - T_w(\mathcal{E} \setminus \mathcal{E}')$ and should satisfy the constraint $1 - T_w(\mathcal{E} \setminus \mathcal{E}') \geq \delta$ to allow $\mathcal{S}$ matching to $\mathcal{E}$. To compute the minimum number l given the above constraint, we simply sum up l entity tokens with the largest weight, such that the total weight of the l entity tokens is larger than δ.

The maximum valid matching length u: According to Proposition 2, no insertion cost and no substitution cost are involved, and the only cost is deletion on the redundant matched text tokens. We denote the valid matched text tokens by $\mathcal{S}'$, and the redundant matched text tokens by $\mathcal{S}''$. The total weight of the valid matched tokens is

$$T_w(\mathcal{S}') = \frac{T_{idf}(\mathcal{S}')}{T_{idf}(\mathcal{S})} \leq \frac{T_{idf}(\mathcal{S}')}{T_{idf}(\mathcal{S}') + T_{idf}(\mathcal{S}'')}.$$

Since the candidate sub-string should match the entity, we have the constraint $T_w(\mathcal{S}') \geq \delta$. So, we have

$$\frac{T_{idf}(\mathcal{S}')}{T_{idf}(\mathcal{S}') + T_{idf}(\mathcal{S}'')} \geq \delta.$$

From Proposition 2, $T_{idf}(\mathcal{S}')$ equals to $T_{idf}(\mathcal{E})$ and is a constant. The number of the tokens in $\mathcal{S}''$ is maximised when each redundant token has the smallest IDF value. Therefore, the maximum length u equals to the number of the tokens of $\mathcal{S}'$ plus the number of tokens in $\mathcal{S}''$ that has all the text tokens match the entity token with the smallest IDF value.

2) Analysis of FuzzyED with the enumeration-based candidate sub-string producing technique: This enumeration-based approach produces many unpromising candidate sub-strings. More precisely, the total number of candidate sub-strings generated is $(u - l) \times k$, where k is the number of matched tokens in the document (i.e. length of the position list). Next, we propose a spanning-based candidate sub-string producing technique that reduces the number of candidate sub-strings to smaller than k.

D. Producing candidate sub-strings by spanning

We present our second approach of producing candidate sub-strings here. The key idea is that we start from a core token and perform left/right spanning to obtain candidate sub-strings. We deduce a lemma that guarantees any sub-strings without a core token will not match to the entity. To determine when the spanning can be terminated, we develop a lower bound dissimilarity based on the similarity function.

We observe that the large number of unpromising candidate sub-strings generated by enumeration is because many matched tokens are not important tokens (i.e. tokens with small IDF values [32]). Those tokens are likely to appear many times in a document and result in generating many unpromising candidate sub-strings. Here, we propose a spanning-based candidate sub-string producing technique that makes use of important tokens which we call *core tokens*. Our spanning-based technique starts from a core token and uses left and right spanning to find candidate sub-strings for the similarity evaluation. To determine when the left spanning and right spanning should stop, we design a lower bound dissimilarity derived from the FuzzyED similarity.

1) Finding core tokens of an entity: As we have discussed in Section II-B, each token is associated with a weight. The weights of tokens can help reduce the number of unpromising candidate sub-strings. In this paper, we use IDF to measure the weights of tokens, although other weighting schemes (e.g. TF-PDF [9]) can be straightforwardly applied in our algorithm. Our key idea is to find a subset of entity tokens (i.e. core tokens) to represent the entity.

Example 2: We can use core tokens {university, melbourne} to represent the entity with tokens {the, university, of, melbourne}. The tokens with smaller weights, such as {the, of}, are called *optional tokens*.

Formally, given an entity similarity threshold δ and an entity with m tokens $\mathcal{E} = \{\mathcal{E}_1, \mathcal{E}_2, ..., \mathcal{E}_m\}$, we construct a set $\mathcal{C}$ of q tokens to represent the entity $\mathcal{E}$ where $\mathcal{C} \subseteq \mathcal{E}$. The remaining $(m - q)$ tokens in $\mathcal{E} \setminus \mathcal{C}$ form a set O corresponding to the optional tokens. The property of core tokens is that at least one core token should appear in a sub-string candidate to allow the candidate to match the entity. The core token set $\mathcal{C}$ satisfies the following constraint.

$$T_w(\mathcal{C}) > 1 - \delta \tag{8}$$

The above constraint guarantees that the total weight of tokens in the optional token set O to be smaller than δ. The following Lemma shows that at least one core token should appear in a candidate sub-string to allow the candidate sub-string to match the entity.

Lemma. *Given a candidate sub-string $\mathcal{S}$ that matches an entity $\mathcal{E}$ (i.e. the similarity between $\mathcal{S}$ and $\mathcal{E}$ is not smaller than δ), the candidate sub-string $\mathcal{S}$ must have at least one text token matching to a core token of the entity $\mathcal{E}$.*

Proof. Suppose no token in $\mathcal{S}$ matches the core tokens of $\mathcal{E}$. For transforming $\mathcal{S}$ to $\mathcal{E}$, at least we need to insert all the core tokens in $\mathcal{E}$ to $\mathcal{S}$ and the total cost of the insertion is larger than $1-\delta$ (cf. Constraint (8)). Hence, the similarity between $\mathcal{S}$ and $\mathcal{E}$ is smaller than δ. Therefore, for $\mathcal{S}$ to match $\mathcal{E}$ (i.e. the similarity between $\mathcal{S}$ and $\mathcal{E}$ is not smaller than δ), at least one text token in $\mathcal{S}$ must match a core token of the entity $\mathcal{E}$. □

According to the above lemma, the candidate sub-strings containing no core tokens can be discarded without sacrificing recall. Hence, core tokens are good starting points to find the candidate sub-strings.

Note that the number of core tokens of an entity should be as small as possible, because a core token may match many text tokens. Those matched text tokens may further generate many candidate sub-strings. To minimise the number of core tokens to represent an entity, we select q tokens with the largest weights from $\mathcal{E}$ to make $\mathcal{C}$ just satisfy Constraint (8).

2) Producing a candidate sub-string via spanning: Since the core tokens represent the entity, we only use the core tokens as query tokens to find their matching positions in the document using Li et al.'s algorithm. The matched results of the entity in the document are similar to the results shown in Figure 1, except that all those matched tokens are core tokens this time. As the boundaries of the candidate sub-strings may not be the core tokens, we need to check the left side and the right side of the core token and see if any optional tokens are near the core token. We use *left spanning* and *right spanning* to find the left boundary and right boundary, respectively. To determine when the spanning should be terminated, we compute a lower bound dissimilarity between a candidate sub-string and the entity. When the left spanning or right spanning results in the lower bound dissimilarity higher than the threshold $(1-\delta)$, the spanning should be terminated.

Figure 4 shows an overview of the process of finding the boundaries of a candidate sub-string. Initially, the candidate sub-string which we call *current candidate sub-string* has only one token (i.e. the core token $\mathcal{C}_1$). Then, the left spanning leads to an optional token O_1 included in the current candidate sub-string. The left spanning is terminated because of the lower bound dissimilarity is higher than $(1-\delta)$ if more tokens in the left side are included. After the right spanning, the current candidate sub-string covers one more core token (i.e. $\mathcal{C}_2$) and one optional token (i.e. O_3). The current candidate sub-string cannot be further extended because of the high lower bound dissimilarity, and hence we obtain the full candidate sub-string which requires computing FuzzyED similarity.

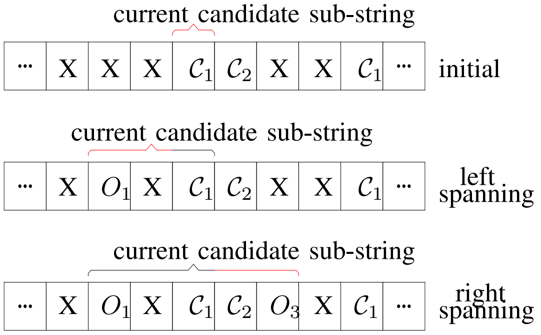
current candidate sub-string
··· X X X $\mathcal{C}_1$ $\mathcal{C}_2$ X X $\mathcal{C}_1$ ···
initial
current candidate sub-string
··· X O_1 X $\mathcal{C}_1$ $\mathcal{C}_2$ X X $\mathcal{C}_1$ ···
left spanning
current candidate sub-string
··· X O_1 X $\mathcal{C}_1$ $\mathcal{C}_2$ O_3 X $\mathcal{C}_1$ ···
right spanning

Fig. 4. Spanning from the core token

Example 3: Given a document $\mathcal{D}$ = "*... located near the city cbd, melbounre univercity, a top university in australia...*" and an entity $\mathcal{E}$ = "the university of melbourne", then the core tokens are {university, melbourne} and the optional tokens are {the, of} according to their IDF values. FuzzyED starts with the core token "melbourne" in the document. After performing left and right spanning, a candidate sub-string "melbourne univercity" is produced. The token "the" before "city cbd" and the second "university" in the document are not included in this candidate sub-string due to the lower bound dissimilarity (discussed next) exceeding the threshold.

The lower bound dissimilarity: As demonstrated in Figure 4, we start with a candidate sub-string with a core token, and then enlarge the candidate sub-string by the left spanning and the right spanning. When we transform a candidate sub-string to the entity, a token of the candidate sub-string $\mathcal{S}$ is either deleted or substituted. Therefore, including a token to the current candidate sub-string involves a cost, i.e. deletion cost or substitution cost. We compute the lowest cost for including each text token. Hence, the sum of the lowest cost of each token in the current candidate sub-string is the lower bound dissimilarity, denoted by $\mathcal{B}_\perp$. Next, we explain how to compute the total deletion cost and the total substitution cost in $\mathcal{B}_\perp$ while fining the candidate sub-string.

To compute the deletion cost in $\mathcal{B}_\perp$ more efficiently, we maintain the total IDF values $\mathcal{V}_T$ for all the tokens in the current candidate sub-string, and the total IDF values $\mathcal{V}_R$ for those text tokens needed to be deleted in the current candidate sub-string. $\mathcal{V}_T$ is initialised to the IDF value of the core token and $\mathcal{V}_R$ is initialised to 0.

The substitution cost between two similar tokens $\mathcal{E}_i$ and $\mathcal{S}_j$ is $(1-eds(\mathcal{E}_i,\mathcal{S}_j)) \times (w(\mathcal{E}_i)+w(\mathcal{S}_j))$ according to Equation (6). As there may exist another not included token $\mathcal{S}_r$ that is more similar to $\mathcal{E}_i$ than $\mathcal{S}_j$, i.e. $eds(\mathcal{E}_i,\mathcal{S}_r) > eds(\mathcal{E}_i,\mathcal{S}_j)$. If $\mathcal{S}_r$ exists, we need to delete $\mathcal{S}_j$ with cost $w(\mathcal{S}_j)$. Note that the substitution cost $(1-eds(\mathcal{E}_i,\mathcal{S}_j)) \times (w(\mathcal{E}_i)+w(\mathcal{S}_j))$ may be larger than the deletion cost $w(\mathcal{S}_j)$. Hence, the lowest cost of including $\mathcal{S}_j$ to the current candidate sub-string is set to $(1-eds(\mathcal{E}_i,\mathcal{S}_j)) \times w(\mathcal{S}_j)$ which is smaller than both $(1-eds(\mathcal{E}_i,\mathcal{S}_j)) \times (w(\mathcal{E}_i)+w(\mathcal{S}_j))$ and $w(\mathcal{S}_j)$. Because we do not know the weight of the text token (i.e. $w(\mathcal{S}_j)$) as we do not know the full candidate sub-string yet (cf. Equation (3)), we use the IDF value of the text token to compute the lower bound. For this reason, the lower bound dissimilarity of including the text token is represented by $(1-eds(\mathcal{E}_i, \mathcal{S}_j)) \times idf(\mathcal{S}_j)$. This substitution cost is equivalent to deleting a token with an IDF value of $(1 - eds(\mathcal{E}_i, \mathcal{S}_j)) \times idf(\mathcal{S}_j)$. Since the substitution cost can be viewed as deletion cost, in what follows we compute the lower bound dissimilarity as if we only considered deletion cost.

For ease of computing the lower bound, we also maintain an array M of length $|\mathcal{E}|$. The i^{th} element of the array M (i.e. M_i) corresponds to the edit similarity between the most similar text token and the i^{th} entity token of $\mathcal{E}$ (i.e. $\mathcal{E}_i$). We compute the lower bound dissimilarity by the following equation.

$$\mathcal{B}_{\perp} = \frac{\mathcal{V}_R + \sum_i (1 - M_i) \times idf(\mathcal{S}'_i)}{\mathcal{V}_T + \sum_r idf(\mathcal{E}_r)} \tag{9}$$

where $i \in \{i : \tau \leq M_i \leq 1\}$ and $r \in \{r : M_r < 1\}$.

The numerator of Equation (9) represents the total "deletion cost": true deletion cost $\mathcal{V}_R$ and the substitution cost which is $\sum_i (1 - M_i) \times idf(\mathcal{S}'_i)$, where $\mathcal{S}'_i$ is the text token that is the most similar to $\mathcal{E}_i$. The denominator is the ideal total IDF value of the candidate sub-string. $\mathcal{V}_T$ is the total IDF value of the current candidate sub-string. The term $\sum_r idf(\mathcal{E}_r)$ of the denominator is the total IDF value of all the not exactly matched entity tokens. The not included text token next to the current candidate sub-string may exactly match the entity token $\mathcal{E}_r$ without any cost (i.e. exact matching). We can prove that the lower bound increases monotonically as the candidate sub-string spans. The key idea of the proof is that adding the same value $idf(t) > 0$ to the numerator and the denominator of Equation (9) leads to the value of $\mathcal{B}_{\perp}$ increasing. The proof is straightforward and hence omitted.

Left spanning: To begin with, we provide the details of finding the left boundary for a candidate sub-string. We start from the first matched text token t (e.g. the first $\mathcal{C}_1$ in Figure 4) in the document. Without loss of generality, we suppose the text token t matches the i^{th} token of the entity $\mathcal{E}$, so we set the i^{th} element of the array M to the edit similarity of t and $\mathcal{E}_i$ (i.e. $M_i = eds(t, \mathcal{E}_i)$). Then, we span to the left side of the current candidate sub-string by one text token, denoted by t'. For updating $\mathcal{V}_T$ and $\mathcal{V}_R$ in this spanning, we need to handle the following two cases separately.

- Case 1: t' does not match any optional tokens of $\mathcal{E}$, so we need to delete t'. Hence, we update $\mathcal{V}_R$ by $\mathcal{V}_R = \mathcal{V}_R + idf(t')$, and we update the total IDF value $\mathcal{V}_T$ by $\mathcal{V}_T = \mathcal{V}_T + idf(t')$.
- Case 2: t' matches an optional token $\mathcal{E}_j$ of $\mathcal{E}$. We update $\mathcal{V}_T$ by $\mathcal{V}_T = \mathcal{V}_T + idf(\mathcal{E}_j)$. We consider this as a substitution operation and we update $\mathcal{V}_R$ in two scenarios.
    - No other text token in the current candidate sub-string matches $\mathcal{E}_j$. We update M_j by $M_j = eds(t', \mathcal{E}_j)$. We do not update $\mathcal{V}_R$, due to no deletion required.
    - A text token in the current candidate sub-string has matched to $\mathcal{E}_j$. We update $\mathcal{V}_R$ by $\mathcal{V}_R = \mathcal{V}_R + idf(\mathcal{S}'_j)$, and M_j by $M_j = eds(t', \mathcal{E}_j)$ if $eds(t', \mathcal{E}_j) > M_j$. Otherwise, we update $\mathcal{V}_R$ by $\mathcal{V}_R = \mathcal{V}_R + idf(t')$.

After $\mathcal{V}_T$ and $\mathcal{V}_R$ are updated, we compute the lower bound $\mathcal{B}_{\perp}$ using Equation (9). If $1 - \mathcal{B}_{\perp} \leq \delta$, we span the current candidate sub-string to cover the token t'. Otherwise, the left spanning is terminated. After the termination, the leftmost matched text token is identified as the left boundary of the candidate sub-string.

Right spanning: After the left spanning, we span the current candidate sub-string to cover the tokens in its right side. Similar to the left spanning, we compute the lower bound using Equation (9). If $1 - \mathcal{B}_{\perp} > \delta$, we terminate the right spanning and the rightmost position of the candidate sub-string is the right boundary.

3) Reusing computation in producing candidate sub-strings: The boundaries of a candidate sub-string should start and end with matched tokens, because the unmatched tokens next to the boundaries are not part of the entity. We can use this property to reuse some computation while finding the boundaries of a neighbour candidate sub-string (i.e. a candidate sub-string next to the previously found candidate sub-string). We refer to the text token that matches an entity token as the *landmark* token.

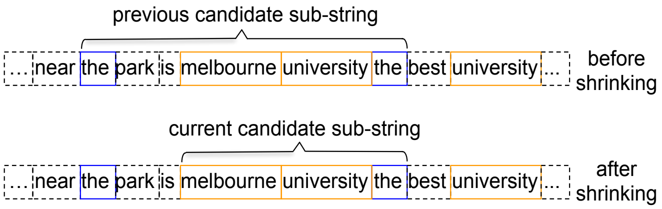
previous candidate sub-string
... near the park is melbourne university the best university ...
before shrinking
current candidate sub-string
... near the park is melbourne university the best university ...
after shrinking

Fig. 5. Shrinking the previous candidate sub-string

Shrinking: To find the neighbour candidate sub-string, we shrink the previous candidate sub-string by one landmark token. That is the left boundary is moved from the leftmost landmark, denoted by l_1, to the second leftmost landmark, denoted by l_2.

Example 4: Given a piece of text "*... near the park is melbourne university the best university in australia*". Suppose the leftmost landmark l_1 and the second leftmost landmark l_2 of the previous candidate sub-string are "*the*" and "*melbourne*", respectively; the leftmost landmark of the candidate sub-string after shrinking is "*melbourne*". Figure 5 gives an example of shrinking the previous candidate sub-string.

Suppose l_1 matches the i^{th} entity token $\mathcal{E}_i$. We update $\mathcal{V}_R$ using the following equation.

$$\mathcal{V}_R = \begin{cases} \mathcal{V}_R - \mathcal{V}_s - idf(l_1) & \text{if } eds(l_1, \mathcal{E}_i) < M_i, \\ \mathcal{V}_R - \mathcal{V}_s - idf(t) & \text{otherwise.} \end{cases} \tag{10}$$

where $\mathcal{V}_s = \sum idf(t_j)$, and t_j is the text token between the leftmost landmark l_1 and the second leftmost landmark l_2; if $eds(l_1, \mathcal{E}_i) = M_i$, we need to update M_i by $M_i = eds(t, \mathcal{E}_i)$

where t is the second most similar token to $\mathcal{E}_i$ in the previous candidate sub-string.

The total IDF value $\mathcal{V}_T$ of the candidate sub-string after shrinking can be updated as follows.

$$\mathcal{V}_T = \mathcal{V}_T - \mathcal{V}_s - idf(l_1) \tag{11}$$

After the shrinking, we can start the right spanning to find the right boundary of the new candidate sub-string.

4) Analysis of FuzzyED with the spanning-based candidate sub-string producing algorithm: In FuzzyED, the number of candidate sub-strings required the similarity evaluation is k at most, where k is the number of matched tokens (including core tokens and optional tokens). To explain this, we refer to Figure 5. Every time, we shrink the previous candidate sub-string by one matched text token and find a new candidate sub-string. Hence, we perform k shrinking at most, and each shrinking corresponds to a candidate sub-string. Therefore, the spanning-based candidate sub-string producing algorithm generates k candidate sub-strings at most. In comparison, the enumeration-based candidate sub-string producing algorithm generates $(u-l)\times k$ candidate sub-strings as we have analysed in Section IV-C2.

The spanning-based candidate sub-string producing algorithm can be applied to the case of not using core tokens. We conduct experiments to investigate the importance of core tokens in Section V.

E. Pruning techniques

Here we describe two pruning techniques. The intuition of the first pruning technique is that if a sub-string requires edition cost exceeding the limit, then the sub-string will not match the entity and can be pruned. The intuition of the second pruning technique is that if the total weight of the matched tokens is less than the minimum value, then the sub-string will not match the entity and can be pruned as well.

The time complexity of computing FuzzyED similarity is $\mathcal{O}(mn)$ for an entity $\mathcal{E}$ and a candidate sub-string $\mathcal{S}$ as we have discussed in Section IV-B1. In the approximate entity extraction problem, the number of entities and candidate sub-strings is large especially for large datasets. To enhance the efficiency of FuzzyED, we should avoid as many similarity computations as possible.

1) A lower bound cost: To reduce the number of similarity computations, we use a technique to prune some candidate sub-strings. The key idea of the pruning technique is to compute **a lower bound cost** for transforming a candidate sub-string to an entity, and is to prune the candidate sub-string with a lower bound cost higher than a certain threshold. The lower bound cost includes insertion and substitution cost, and is computed by the equation below.

$$C_{\perp}(\mathcal{E}, \mathcal{S}) = \sum_{\mathcal{E}_i \in \mathcal{E}} (1 - M_i) \times w(\mathcal{E}_i) \tag{12}$$

where M_i is the edit similarity of the entity token $\mathcal{E}_i$ to the most similar text token in the candidate sub-string $\mathcal{S}$. Note that both the insertion cost and the substitution cost are considered in the above equation, because $M_i = 0$ is the case of insertion and $M_i > 0$ is the case of substitution. If the lower bound cost $C_{\perp}(\mathcal{E}, \mathcal{S})$ is higher than the threshold $1-\delta$, we prune the candidate sub-string and avoid computing $FuzzyEDS(\mathcal{E}, \mathcal{S})$.

2) A general filtering technique: We notice that a filtering technique introduced by Chakrabariti et al. [6] can be used in FuzzyED. For completeness, we use the filter to improve the efficiency of FuzzyED. Formally, a sub-string candidate can be pruned if the condition below is satisfied.

$$T_w(\mathcal{S} \cap \mathcal{E}) < \delta \tag{13}$$

We explain the intuition of the above filter in the following. If the total weight of the matched tokens in the sub-string candidate is smaller than δ, then the sub-string candidate will not match to the entity and hence can be pruned. $T_w(\mathcal{S} \cap \mathcal{E})$ can be viewed as the upper bound score of $\mathcal{S}$ matching to $\mathcal{E}$. The sub-string candidate is filtered out if its upper bound score is smaller than the similarity threshold. The filter helps reduce the cost of further computing the token-level edit-distance.

V. Experimental Study

In this section, we present the empirical results on the efficiency and effectiveness of FuzzyED denoted by "FED" in comparison with a baseline denoted by "FJ" discussed in Section V-A2.

A. Experiment Setup

All experiments were conducted on a machine running Linux with an Intel Xeon E5-2643 CPU and 32GB memory. We set the entity similarity threshold δ to 0.9, and the token edit similarity threshold τ to 0.8.

1) Datasets: We used six publicly available real-world datasets in our experiments. They are Amazon Reviews [33], DBWorld Messages, IMDB Reviews [34], News, Tweet and Wikipedia articles. The details of the datasets are as follows.

- Amazon Reviews: the dataset contains 346,867 product reviews from the customers of Amazon. Each product review serves as a document; 1,989,376 product names from Amazon form the entity dictionary.
- DBWorld Messages: the dataset contains 33, 628 messages of "call for papers", job advertisement and so forth in the database research community. Each message is a document; the entity dictionary contains 132,745 worldwide institution names from Free-base [35].
- IMDB Reviews: the dataset has 97,788 movie reviews from the IMDB website. Each movie review is a document; the entity dictionary contains 108,941 movie names in the IMDB website.
- News: the dataset contains 90,328 news [36] and 3,199,972 company names which was used to form the entity dictionary. The company names were downloaded from the website of Australian Securities and Investments Commission[2].

[2] https://goo.gl/7Uvvo3

TABLE II
DETAILS OF DOCUMENTS AND DICTIONARIES

dataset	size	ave. len.	max. len.	min. len.
Amazon doc.	346,867	191	29,070	30
Amazon dict.	1,989,376	6	204	1
DBWorld doc.	33,628	732	33,648	1
DBWorld dict.	132,745	3	27	1
IMDB doc.	97,788	277	2,968	8
IMDB dict.	108,941	3	24	1
News doc.	90,328	49	277	3
News dict.	3,199,972	4	28	1
Tweet doc.	14,873	54	90	1
Tweet dict.	7,100,397	2	17	1
Wikipedia doc.	1,273	189,595	207,250	159,585
Wikipedia dict.	20,483	2	6	1

- Tweet: this dataset contains 14,873 tweets[3], and 7,100,397 celebrity names from the IMDB website were used to form the entity dictionary.
- Wikipedia: the dataset has 1,273 articles downloaded from Wikipedia[4], and the entity dictionary contains 20,483 street names obtained from OSNI Open Data[5].

More details of the datasets are provided in Table II; the average, maximum and minimum length of the documents or the entities in the dictionary were measured by the number of tokens.

2) The extended Fuzzy Jaccard algorithm: As we have discussed in Section III-B, Fuzzy Jaccard [30] can be extended to extract entities when it equips with our proposed candidate sub-string producing techniques. Formally, Jaccard similarity of $\mathcal{E}$ and $\mathcal{S}$ is defined as follows.

$$JAC = \frac{|\mathcal{E} \cap \mathcal{S}|}{|\mathcal{E} \cup \mathcal{S}|} = \frac{|\mathcal{E} \cap \mathcal{S}|}{|\mathcal{E}| + |\mathcal{S}| - |\mathcal{E} \cap \mathcal{S}|} \tag{14}$$

where $|\mathcal{E} \cap \mathcal{S}|$ is the number of matched tokens between $\mathcal{E}$ and $\mathcal{S}$; $|\mathcal{E} \cup \mathcal{S}|$ is the number of tokens in the union of $\mathcal{E}$ and $\mathcal{S}$; $|\mathcal{E}|$ and $|\mathcal{S}|$ are the number of tokens in $\mathcal{E}$ and $\mathcal{S}$, respectively. Due to the tolerance of character mismatching, one text token may match multiple tokens of an entity and vice versa. In entity extraction applications, an entity token can match at most one token of a candidate sub-string and vice versa. Following the settings of the original Fuzzy Jaccard algorithm, the maximum weight matching algorithm [37] is applied to guarantee the one to one matching in Fuzzy Jaccard. By considering the weights of tokens (cf. Section II-B), we can write Fuzzy Jaccard similarity as follows.

$$FJ = \frac{\frac{1}{2}\sum_{e \in \mathcal{E}', s \in \mathcal{S}'} eds(e,s) \cdot (w(e) + w(s))}{1 + 1 - \frac{1}{2}\sum_{e \in \mathcal{E}', s \in \mathcal{S}'} eds(e,s) \cdot (w(e) + w(s))} \tag{15}$$

For a fairer comparison, we adapt all the FuzzyED's techniques to the extended Fuzzy Jaccard with calibration specifically for the Jaccard similarity. The filter explained in Section IV-E2 proposed by Chakrabariti et al. [6] is also used in the extended Fuzzy Jaccard. Please refer to our technical report[6] for more details about the adaptation of the other techniques.

3) Implementations: We have implemented four algorithms in C++: FED-e (resp. FED-s) is FuzzyED together with the enumeration-based (resp. spanning-based) candidate producing technique; FJ-e (resp. FJ-s) is Fuzzy Jaccard together with the enumeration-based (resp. spanning-based) candidate producing technique.

TABLE III
OVERALL EFFICIENCY COMPARISON

sub-dataset	using enumeration		using spanning	
	FED-e	FJ-e	FED-s	FJ-s
Amazon	1.05 h	26.7 h	7 sec	10 sec
DBWorld	0.25 h	12.9 h	11 sec	11 sec
IMDB	0.13 h	6.06 h	11 sec	12 sec
News	0.7 h	1.1 h	18 sec	26 min
Tweet	44 sec	49 sec	2 sec	12 sec
Wikipedia	0.47 h	0.75 h	2.3 min	3.9 min

B. Efficiency and effectiveness comparison

Here, we investigate the performance of our algorithm in three aspects: overall efficiency, the effect of varying the parameters (e.g. τ and δ) on the efficiency, and effectiveness.

1) Overall efficiency: We conducted experiments on the six datasets for FED-s and the elapsed time of FED-s for Amazon Reviews, DBWorld Messages, IMDB Reviews, News, Tweet and Wikipedia. is 16 hours, 17 minutes, 16 minutes, 31 minutes, 3 minutes and 6.2 hours, respectively; FJ-s took *twice more* time than FED-s to process the datasets. In comparison, FED-e and FJ-e were extremely slow to process the whole datasets, because they require measuring the two-level similarity for more sub-string candidates as discussed in Section IV-C2. To provide some specific results on the elapsed time of the four implementations, we randomly sampled a sub-dataset from each of the original document dataset. To construct the six sub-datasets, we sampled 1 per 100 documents in DBWorld Messages, IMDB Reviews, News, Tweet and Wikipedia, and 1 per 10,000 documents in Amazon Reviews. Thus, FJ-e and FED-e can process the sub-datasets in a reasonable amount of time.

Table III gives the efficiency of the four implementations on the six sub-datasets. As we see from the table, implementations using spanning-based candidate producing technique (i.e. FED-s and FJ-s) are often 10x times faster than those using enumeration-based candidate producing technique. Another observation is that FED-s is slightly more efficient than FJ-s, because the dataset is too small to demonstrate the advantage of FED. Recall that FED-s is two times faster than FJ-s when using the whole dataset.

2) Effect of varying the parameters on efficiency: Next, we study the effect of varying the parameters on the efficiency of FED-s, FED-e, FJ-s and FJ-e. We used the DBWorld Messages dataset as a representative in this set of experiments. The default settings of the experiments are as follows: (i) the

[3] https://goo.gl/MFCTcr

[4] https://dumps.wikimedia.org/enwiki/latest/

[5] https://goo.gl/GSQAUs

[6] https://arxiv.org/pdf/1702.03519.pdf

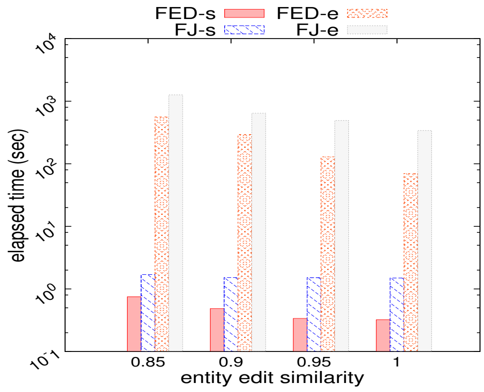

Fig. 6. Varying δ

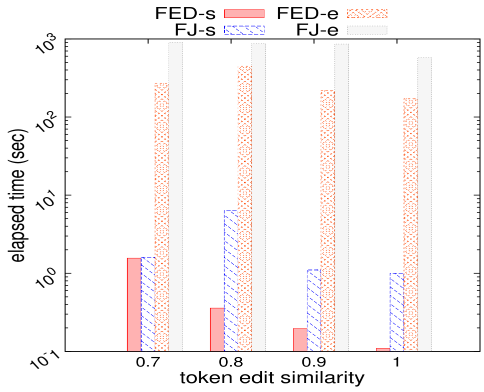

Fig. 7. Varying τ

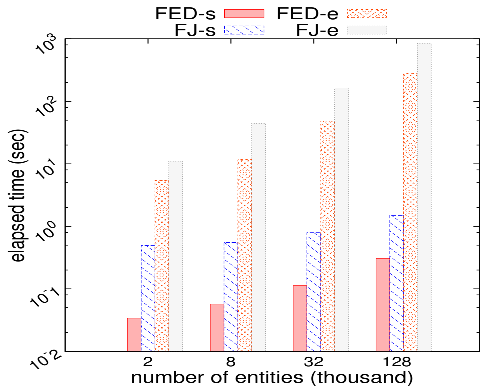

Fig. 8. Varying dictionary size

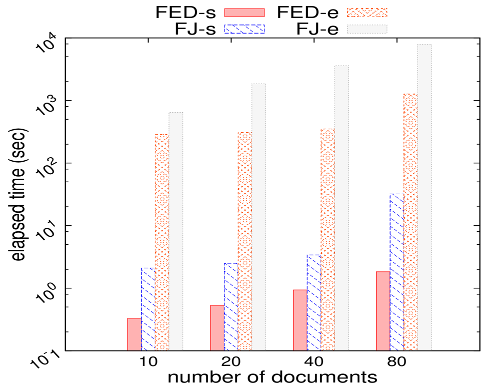

Fig. 9. Varying # of doc.

entity similarity threshold δ was set to 0.9; (ii) the token edit similarity threshold τ was set to 0.8; (iii) the number of entities in the dictionary is 132,745 (i.e. the whole dictionary) and (iv) the number of documents is 10.

Effect of varying the entity similarity threshold: To study the effect of the entity similarity threshold δ, we varied δ from 0.85 to 1. Figure 6 shows the results of the effect on the four implementations. As can be seen from the figure, FED based implementations consistently outperform FJ based implementations. Implementations using spanning-based candidate producing technique outperform those using enumeration-based candidate producing technique by around 100 times. An observation of the figure is that as the entity similarity threshold decreases the total elapsed time of all the implementations increases. This is because when the entity similarity threshold is small, more candidates require measuring the two-level similarity.

Effect of varying the token similarity threshold: Figure 7 gives the results of varying the token similarity threshold τ from 0.7 to 1. FED-s and FJ-s significantly outperform FED-e and FJ-e by two orders of magnitude. Similar to varying δ, the smaller the threshold, the more time our algorithm requires.

Effect of varying the size of the entity dictionary: To study the effect of the size of the dictionary, we varied the number of entities in the dictionary from 2,000 to 128,000. Figure 8 shows that the elapsed time of all the four implementations increases as the size of the dictionary increases.

Effect of varying the number of documents: To study the effect of the number of documents on the efficiency, we sampled from the DBWorld Messages dataset four sub-datasets of 10, 20, 40 and 80 documents with the average length of 732. We measured the total elapsed time of extracting entities from each sub-dataset. As shown in Figure 9, the elapsed time of FED based implementations grows more slowly compared with FJ based ones. This is because the more documents, the more sub-string candidates are generated. As a result, our algorithm needs to measure more two-level similarity. The elapsed time of FJ based implementations increases faster than that of FED based ones.

3) Overall effectiveness: To demonstrate the effectiveness of the FuzzyED similarity and the Fuzzy Jaccard similarity, we used the whole dataset of DBWorld Messages. We manually labelled 20,000 sub-string candidates as a set of ground truth. Entities in the document correctly extracted as entities in the dictionary are called true positive (denoted by tp); no entities in the document extracted as entities in the dictionary are called false positive (denoted by fp). We compute the precision p and recall r by $p = \dfrac{tp}{tp + fp}$ and $r = \dfrac{tp}{tp + fn}$, where fn is the number of false negative and hence $tp + fn$ is the total number of true positives in the ground truth set.

Table IV shows the results of F-measure for FED and FJ

TABLE IV
F-MEASURE OF FED AND FJ

δ	precision		recall		F_1	
	FED	FJ	FED	FJ	FED	FJ
1.00	100%	97.6%	94.5%	95.4%	97.2%	96.5%
0.95	88.0%	85.3%	94.8%	95.5%	91.3%	88.4%
0.90	71.5%	69.5%	96.6%	97.1%	82.2%	81.0%
0.85	64.1%	62.6%	99.7%	100%	78.0%	77.0%

TABLE V
EFFICIENCY OF CORE TOKENS ON CANDIDATES PRODUCING

sub-dataset	FED-s	FED-a	speedup
Amazon	7 sec	0.70 h	362
DBWorld	11 sec	0.14 h	45
IMDB	11 sec	0.17 h	56
News	18 sec	5.1 h	1036
Tweet	2 sec	2.2 min	66
Wikipedia	2.3 min	48 h	1250

on the entity similarity threshold δ changing from 0.85 to 1. As we can see from the table, FED has better F_1 score and precision than FJ and comparable recall to FJ, since FJ totally ignores order information of tokens in an entity.

The percentage of occurrence of the problems where both typos and name variations occurred in our labeled candidate sub-strings is about 2‰. Here are three examples.

- The sub-string in text is *university of hong kong of sciences and technology* which matches to the entity *hong kong university of science and technology* in the dictionary. The matching score for this pair is 0.919759.
- The sub-string in text is *universitat rovira i virgili* which matches to the entity *rovira i virgili university* in the dictionary. We have found that "universitat" is the spelling of "university" in Spain. The matching score for this pair is 0.915718.
- The sub-string in text is *universite claude bernard lyon* which matches to the entity *claude bernard university lyon* in the dictionary. We have found that "universite" is the spelling of "university" in French. The matching score for this pair is 0.922923.

C. Effect of core tokens

In this set of experiments, we provide experimental results of the spanning-based approach using core tokens compared with the spanning-based approach without using core tokens as discussed in Section IV-D4. The datasets used in these experiments are identical to those detailed in Table III.

To demonstrate the effectiveness of using core tokens, we used two versions of FED: one with core tokens applied in the candidate producing process; the other, denoted by FED-a ("a" for all entity tokens), without using core tokens in the candidate producing process. Note that the FJ based approach without using core tokens were extremely slow and did not complete within our time limit, and hence the results of FJ are not shown here. As we can see from Table V, FED-s consistently outperforms FED-a by orders of magnitude. This is because using core tokens reduces the number of matched tokens in the document, and hence significantly reduces the number of sub-string candidates produced.

TABLE VI
PRECISION, RECALL AND F_1 SCORE COMPARISON OF FED-S AND FED-A

δ	precision		recall		F_1	
	FED-s	FED-a	FED-s	FED-a	FED-s	FED-a
1.00	100%	100%	94.5%	94.5%	97.2%	97.2%
0.95	88.0%	88.0%	94.8%	94.8%	91.3%	91.3%
0.90	71.5%	71.5%	96.6%	96.6%	82.2%	82.2%
0.85	64.1%	64.1%	99.7%	99.7%	78.0%	78.0%

TABLE VII
COMPARISON WITH THE ONE-LEVEL EDIT-DISTANCE BASED APPROACH

algorithm	Amazon	DBWorld	IMDB	News	Tweet	Wikipedia
TASTE	>5 days	2.7 h	9.5 h	11 h	7.7 h	4.5 h
FED	16 h	17 min	16 min	31 min	3 min	3.2 h

We have conducted experiments to compare the effectiveness of FED-s and FED-a to study the effectiveness of core tokens. The experimental results are shown in Table VI. The results show that the effectiveness (measured by precision, recall and F_1 score) of FED-s and FED-a is identical. This is because the technique of core tokens can be treated as a pruning technique, where only the sub-strings that will not match to any entity are pruned.

D. Comparison with one-level edit-distance based approach

Here we compare our FuzzyED with the one-level edit-distance based approach. Since the approach called *TASTE* outperforms that proposed by Wang et al. [4] as shown in the experiments [5], we compared our algorithm against TASTE. As we can see from Table VII, FuzzyED consistently outperforms TASTE by an order of magnitude, thanks to our series of novel techniques (e.g., core tokens, spanning and pruning techniques) to improve efficiency discussed in Section IV-D and Section IV-E. In terms of effectiveness, FED achieves about 3% higher F_1 score than TASTE on our manually labeled sub-strings discussed in Section V-B3. TASTE cannot recognize *university of hong kong of sciences and technology* as the entity *hong kong university of science and technology* due to the large dissimilarity measured in TASTE [5].

It is worthy to point out that the elapsed time of FuzzyED on the *Amazon* dataset is 16 hours, which seems to be long at first glance. This amount of time is fairly short for the dataset which has nearly two million entities in the dictionary and about half a million documents.

VI. CONCLUSION

In this paper, we have proposed a two-level edit-distance based algorithm (which we call FuzzyED) for the approximate entity extraction. FuzzyED natively allows domain experts to customise the tolerance thresholds of the two levels individually. FuzzyED is efficient thanks to various novel techniques we have proposed. The key techniques include (i) a spanning-based candidate sub-string producing technique, (ii) a lower bound dissimilarity to determine the boundaries of candidate sub-strings, (iii) a core token based technique that makes use of the importance of tokens to reduce the number of unpromising candidate sub-strings, and (iv) a shrinking technique to reuse computation. A comprehensive empirical study has confirmed that our algorithm based on two-level edit-distance is efficient and also effective. We hope that FuzzyED would provide data mining practitioners a better alternative towards named entity extraction from free text.

Acknowledgements

This work is supported by Australian Research Council (ARC) Discovery Project DP180102050. Majority of the work was done when Zeyi Wen was with The University of Melbourne. We also like to thank the anonymous reviewers for their insightful comments.

References

[1] L. Derczynski, D. Maynard, G. Rizzo, M. van Erp, G. Gorrell, R. Troncy, J. Petrak, and K. Bontcheva, "Analysis of named entity recognition and linking for tweets," *Information Processing and Management*, vol. 51, no. 2, pp. 32–49, 2015.

[2] W. Shen, J. Wang, and J. Han, "Entity linking with a knowledge base: issues, techniques, and solutions," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 2, pp. 443–460, 2015.

[3] D. Nadeau and S. Sekine, "A survey of named entity recognition and classification," *Lingvisticae Investigationes*, vol. 30, no. 1, pp. 3–26, 2007.

[4] W. Wang, C. Xiao, X. Lin, and C. Zhang, "Efficient approximate entity extraction with edit distance constraints," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2009, pp. 759–770.

[5] D. Deng, G. Li, and J. Feng, "An efficient trie-based method for approximate entity extraction with edit-distance constraints," in *Proceedings of the IEEE International Conference on Data Engineering*, 2012, pp. 762–773.

[6] K. Chakrabarti, S. Chaudhuri, V. Ganti, and D. Xin, "An efficient filter for approximate membership checking," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2008, pp. 805–818.

[7] K. Zhang and D. Shasha, "Simple fast algorithms for the editing distance between trees and related problems," *SIAM Journal on Computing*, vol. 18, no. 6, pp. 1245–1262, 1989.

[8] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to information retrieval*. Cambridge University Press, 2008.

[9] K. K. Bun and M. Ishizuka, "Emerging topic tracking system," in *The Third International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems*, 2001, pp. 2–11.

[10] G. Navarro, "A guided tour to approximate string matching," *ACM Computing Survey*, vol. 33, no. 1, pp. 31–88, 2001.

[11] G. Li, D. Deng, and J. Feng, "Faerie: efficient filtering algorithms for approximate dictionary-based entity extraction," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2011, pp. 529–540.

[12] A. Gattani, D. S. Lamba, N. Garera, M. Tiwari, X. Chai, S. Das, S. Subramaniam, A. Rajaraman, V. Harinarayan, and A. Doan, "Entity extraction, linking, classification, and tagging for social media: a wikipedia-based approach," *Proceedings of the VLDB Endowment*, vol. 6, no. 11, pp. 1126–1137, 2013.

[13] M.-S. Kim, K.-Y. Whang, J.-G. Lee, and M.-J. Lee, "N-gram/2l: a space and time efficient two-level n-gram inverted index structure," in *Proceedings of the VLDB Endowment*, 2005, pp. 325–336.

[14] Y. Kim and K. Shim, "Efficient top-k algorithms for approximate substring matching," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2013, pp. 385–396.

[15] P. Wang, C. Xiao, J. Qin, W. Wang, X. Zhang, and Y. Ishikawa, "Local similarity search for unstructured text," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2016, pp. 1991–2005.

[16] M. Hadjieleftheriou and D. Srivastava, "Weighted set-based string similarity," *IEEE Data Engineering Bulletin*, vol. 33, no. 1, pp. 25–36, 2010.

[17] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani, "Robust and efficient fuzzy match for online data cleaning," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2003, pp. 313–324.

[18] H. V. Jagadish, B. C. Ooi, K.-L. Tan, C. Yu, and R. Zhang, "idistance: An adaptive b+-tree based indexing method for nearest neighbor search," *ACM Transactions on Database Systems (TODS)*, vol. 30, no. 2, pp. 364–397, 2005.

[19] W. W. Cohen, P. D. Ravikumar, and S. E. Fienberg, "A comparison of string distance metrics for name-matching tasks," in *KDD Workshop on Data Cleaning and Object Consolidation*, vol. 3, 2003, pp. 73–78.

[20] A. Pappu, R. Blanco, Y. Mehdad, A. Stent, and K. Thadani, "Lightweight multilingual entity extraction and linking," in *Proceedings of the ACM International Conference on Web Search and Data Mining*, 2017, pp. 365–374.

[21] X. Carreras, L. Marquez, and L. Padró, "Named entity extraction using AdaBoost," in *Proceedings of the Conference on Natural Language Learning*, vol. 20, 2002, pp. 1–4.

[22] A. Jain and M. Pennacchiotti, "Open entity extraction from web search query logs," in *International Conference on Computational Linguistics*, 2010, pp. 510–518.

[23] W. W. Cohen and S. Sarawagi, "Exploiting dictionaries in named entity extraction: combining semi-markov extraction processes and data integration methods," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2004, pp. 89–98.

[24] M. Yu, G. Li, D. Deng, and J. Feng, "String similarity search and join: a survey," *Frontiers of Computer Science*, vol. 10, no. 3, pp. 399–417, 2016.

[25] Y. Zhang, J. Qi, R. Zhang, and C. Yin, "Pubse: A hierarchical model for publication extraction from academic homepages," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018, pp. 1005–1010.

[26] N. Koudas, B. C. Ooi, K.-L. Tan, and R. Zhang, "Approximate nn queries on streams with guaranteed error/performance bounds," in *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*. VLDB Endowment, 2004, pp. 804–815.

[27] J. Wang, J. Feng, and G. Li, "Trie-join: efficient trie-based string similarity joins with edit-distance constraints," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 1219–1230, 2010.

[28] J. Feng, J. Wang, and G. Li, "Trie-join: a trie-based method for efficient string similarity joins," *The VLDB Journal*, vol. 21, no. 4, pp. 437–461, 2012.

[29] W. Tao, D. Deng, and M. Stonebraker, "Approximate string joins with abbreviations," *Proceedings of the VLDB Endowment*, vol. 11, no. 1, pp. 53–65, 2017.

[30] J. Wang, G. Li, and J. Fe, "Fast-join: an efficient method for fuzzy token matching based string similarity join," in *Proceedings of the IEEE International Conference on Data Engineering*, 2011, pp. 458–469.

[31] J. Wang, G. Li, and J. Feng, "Extending string similarity join to tolerant fuzzy token matching," *ACM Transactions on Database Systems*, vol. 39, no. 1, pp. 1–45, 2014.

[32] I. S. Dhillon, "Co-clustering documents and words using bipartite spectral graph partitioning," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2001, pp. 269–274.

[33] J. McAuley and J. Leskovec, "Hidden factors and hidden topics: understanding rating dimensions with review text," in *ACM Conference on Recommender Systems*, 2013, pp. 165–172.

[34] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Association for Computational Linguistics: Human Language Technologies*, 2011, pp. 142–150.

[35] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor, "Freebase: a collaboratively created graph database for structuring human knowledge," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2008, pp. 1247–1250.

[36] C. Chelba, T. Mikolov, M. Schuster, Q. Ge, T. Brants, P. Koehn, and T. Robinson, "One billion word benchmark for measuring progress in statistical language modeling," *arXiv preprint arXiv:1312.3005*, 2013.

[37] D. B. West, *Introduction to graph theory*. Prentice Hall Upper Saddle River, 2001.