2019 IEEE 35th International Conference on Data Engineering (ICDE)

Scalable Similarity Joins of Tokenized Strings

Ahmed Metwally
The Anti-Abuse Team
LinkedIn Corp.
Mountain View, CA
ametwally@linkedin.com

Chun-Heng Huang
The Ad Traffic Quality Team
Google Inc.
Mountain View, CA
chunheng@google.com

***Abstract*—This work tackles the problem of fuzzy joining of strings that naturally tokenize into meaningful substrings, e.g., full names. Tokenized-string joins have several established applications in the context of data integration and cleaning. This work is primarily motivated by fraud detection, where attackers slightly modify tokenized strings, e.g., names on accounts, to create numerous identities that she can use to defraud service providers, e.g., Google, and LinkedIn.**

To detect such attacks, all the accounts are pair-wise compared, and the resulting similar accounts are considered suspicious and are further investigated. Comparing the tokenized-string features of a large number of accounts requires an intuitive tokenized-string distance that can detect subtle edits introduced by an adversary, and a very scalable algorithm. This is not achievable by existing distance measure that are unintuitive, hard to tune, and whose join algorithms are serial and hence unscalable.

We define a novel intuitive distance measure between tokenized strings, *Normalized Setwise Levenshtein Distance* ($NSLD$). To the best of our knowledge, NSLD is the first metric proposed for comparing tokenized strings. We propose a scalable distributed framework, *Tokenized-String Joiner* (TSJ), that adopts existing scalable string-join algorithms as building blocks to perform $NSLD$-joins. We carefully engineer optimizations and approximations that dramatically improve the efficiency of TSJ. The effectiveness of the TSJ framework is evident from the evaluation conducted on tens of millions of tokenized-string names from Google accounts. The superiority of the tokenized-string-specific TSJ framework over the general-purpose metric-spaces joining algorithms has been established.

I. Introduction

Today, a large number of Internet services are provided to the public, including web-search (e.g., Google Search), online maps (e.g., Google Maps), online product reviews (e.g., Amazon Customer Reviews), online social and professional networks (e.g., Facebook and LinkedIn), video streaming (e.g., YouTube), and sharing rides and residences (e.g., Uber and Airbnb). The high cost of designing, deploying, and maintaining these services is covered by commissions from Internet advertising (e.g., Google, YouTube and Facebook), commissions from e-commerce transactions (e.g., Amazon), members' subscriptions (e.g., LinkedIn), or claiming a share in the sharing economy (e.g., Uber and Airbnb).

For these providers to thrive, it is crucial for them to guarantee high standards on the quality of the offered services. Google, YouTube and Facebook need to protect the Return-On-Investment of their advertisers by ensuring the ads are *viewed* and *clicked* by real surfers, which respectively reflect real exposure to the market and genuine interest in the advertised products. To maintain the credibility of its product reviews, Amazon should show only reviews from real buyers on the listed products. LinkedIn should ensure the profiles targeted by the recruiters represent real professionals. Uber and Airbnb should ensure ride and residence sharing jeopardizes neither the safety of the hosts nor the guests.

For these providers to guarantee high-quality services, it is of utmost importance to foster an environment of trust with their users. This entails identifying ill-intentioned users and fraudsters who disguise as legitimate users. Failure to identify abusive users may hurt the company's image and opportunities [29], [63]. A significant body of recent research has focused on identifying abusive users. Malicious applications on social networks were studied in [49]. Classifying social networks' accounts based on the novelty of their content was proposed in [65]. The analysis of the edges in the social network graphs was used for detecting fake accounts [6], [65], [72], and for mitigating bought online reviews [34]. Limiting the sign-up of fake accounts was studied in [62]. Detection of account takeover and cloning that are then used by the attackers in malicious activities was studied in [5], [20], [21], [32]. Other research focused on identifying the activity/traffic generated by abusive users. Filtering abusive ads traffic was studied in [15], [16], [41], [44], [46], [58], and fake likes and promotion on social networks was studied in [17], [28].

Targeting the aforementioned *silo* attacks and their traffic is a first line defense in the arms race between the service providers and the abusers. However, in the online world, an abuser can have numerous identities[1]. Hence, silo-attack defenses do not safeguard against attacks that are carried out by attackers controlling numerous identities [2], [43]. These identities may not necessarily be active at the same time. These *multi-identity* attacks can be classified roughly into *serial* and *parallel* attacks, or combinations of both[2].

A serial attacker is one who abuses the service provider using a new account every time her old account is terminated.

Part of this work was done while the first author was with Google Inc. The authors like to thank Theodore Hwa of LinkedIn for his insightful discussions and revising the theoretical foundations.

[1]We will use "Identity" and "account" synonymously.

[2]In reputation systems, similar notions to those of serial and parallel attacks are known as *Whitewashing* and *Sybil* attacks, respectively [27], [74].

2375-026X/19/$31.00 ©2019 IEEE
DOI 10.1109/ICDE.2019.00193

These attackers gain experience with every account termination, and work on reverse-engineering the silo-attack defenses.

A parallel attacker, on the other hand, is one who controls numerous accounts with the service providers that are active at the same time. The goal is to launch a sophisticated attack that causes significant damage by driving little abuse from each account, such that each individual account stays under the radar level of the silo-attack defenses. Examples of the malicious or criminal activity conducted using *rings* of fake or hijacked accounts include click fraud [3], [54], content scraping [23], [55], posting fake reviews [14], [33], fake video views [57], and even fake academic paper reviews [50].

Seminal research on multi-identity attacks was conducted in context of click fraud [42], and email spam [75]. The work in [7], [45], [59], [66], [68] clusters the accounts to detect rings of abusive accounts. The recent work in [47], [70] extended this clustering approach using multiple signals simultaneously. The underground market at which abusers buy fake accounts in the thousands was studied in [60].

A. The Motivating Application

This work is motivated by detecting advertising fraud rings. The online ad industry generated $88 billion in 2017 in the U.S. alone [4]. In online advertising, an online-content publisher registers its web sites with the network operator, e.g., Google, to display ads on her sites. A publisher receives revenue to a payment instrument, e.g., bank account, for actions on the displayed ads, e.g., views or clicks by surfers.

Sophisticated fraudsters typically sign up for numerous accounts to spread their revenue and circumvent the silo-attack defenses, which is against the policy of most reputable networks [24]. This motivates clustering publisher accounts for discovering potential click fraud rings [7], [42], [45], [68]. Each publisher account has meta-data, i.e., account attributes. For each account attribute, all the accounts are compared pairwise. The pairs of accounts that are highly similar are used to form edges in a similarity graph for that attribute, where the graph nodes represent accounts. The graph is clustered. The detected clusters flag potential rings.

Some of these attributes are difficult to fake, and are hence excellent candidates for account clustering. One example is the name on the bank account to which the ad-traffic revenue is deposited. Currently, most banks around the world would, if a drastic mismatch is detected in the beneficiary name, not automatically credited the money to the account until manually checked by a bank officer. Providing a fake name to Google would result in the bank rejecting the transfer of the ad-traffic revenue from Google to the attacker's bank account, which defeats the purpose of the attack.

Opening bank accounts incur overhead for fraudsters [40]. Hence, they do not control an infinite number of bank account holders, and strive to maximize their utilization of the resources they control. A Fraudster would try to use the same resource multiple times by slightly altering its form.

For instance, a bank account holder whose name is "Barak Obama" can open multiple bank accounts, and multiple accounts with a service provider under the slightly-edited names "Obamma, Boraak H." or "Burak Ubama". When receiving funds to the right bank account number but to a slightly-edited name, the bank officers would not be alarmed. However, these minor well-crafted edits would circumvent the multi-identify attack defenses of the service provider. If the attack defenses employ naïve tokenized-string comparison techniques, they would not identify these accounts with the service provider as a fraud ring since they have different bank account names.

The bank account holder signal, among other string signals, warrant devising a very scalable technique for comparing tokenized strings in a fuzzy way. Other abuse-detection applications that benefit from string-comparison include detecting paid reviews, detecting fake comments and harassing messages on social networks, and detecting fake job descriptions on professional networks. Moreover, several well-established applications of data integration and cleaning can benefit from comparing tokenizable strings. Such applications include record joining and deduplication in data warehouses, and comparison shopping search engines [22].

As reviewed in Sec. IV, existing tokenized-string comparison algorithms are serial, and hence cannot scale to self-joining tens of millions of records. These algorithms employ distance measures that are sensitive to the order of the tokens (FMS [10]), asymmetric (FMS and AFMS [10]), or are hard to tune since they require setting multiple independent thresholds (all the measures surveyed and proposed in [13], [67]).

B. Our Contributions

Our main contribution can be summarized as follows.

1) In Sec. II, we formulate the problems of fuzzy joins of tokenized strings. Specifically, we define transformations that capture modifications to tokenized strings. We leverage these transformations to define a novel distance between tokenized strings, *Normalized Setwise Levenshtein Distance* ($NSLD$). $NSLD$ is an intuitive metric that is general enough to be applicable to abuse-detection applications as well as the well-established applications of data integration and cleaning. We prove $NSLD$ is a metric, and hence can be leveraged in all flavors of K-nearest-neighbor queries on metric spaces.
2) In Sec. III, we devise *Tokenized-String Joiner* (TSJ), a scalable $NSLD$-joining framework that adopts existing string-join algorithms as building blocks. We carefully engineer optimizations and approximations that trade efficiency for recall.
3) TSJ is generalizable to several parallelizing paradigms, such as MPI and OpenMP. We report the impressive scalability and efficiency of MapReduce-distributed TSJ on tens of millions of names on Google accounts in Sec. V. We discuss the optimizations and the tradeoffs of the approximations, and demonstrate TSJ's superiority over state-of-the-art generic distance-metric algorithms.

II. Problem Statement and Formulation

The aforementioned applications revolve around tokenized strings, i.e., strings that naturally tokenize into substrings

that are meaningful to humans, where slightly editing and shuffling these tokens do not change the interpretation of the string drastically. While it is challenging to mimic the human ability to compare text, a practicable distance/similarity measure to compare tokenized strings is devised. This measure is a metric, and hence can be leveraged in all flavors of K-nearest-neighbor queries on metric spaces, e.g., [12], [48], [61]. Most importantly, it possess the properties that facilitate devising a scalable tokenized-string-joining algorithm[3].

A. Notation and Definition

A distance $D(\cdot,\cdot)$ over a set A is a function that maps a pair of elements in A to a non-negative real number. A distance $D(\cdot,\cdot)$ is a metric if the following holds $\forall \alpha,\beta,\gamma \in A$.

1) Identity: $D(\alpha,\alpha)=0$,
2) Symmetry: $D(\alpha,\beta)=D(\beta,\alpha)$, and
3) Triangle Inequality: $D(\alpha,\beta)+D(\beta,\gamma)\geq D(\alpha,\gamma)$.

Let Σ be a finite alphabet and Σ^* be a set of all finite-length strings over Σ. Denote a string $x\in\Sigma^*$ as $x_1x_2\dots x_{|x|}$, where $|x|$ is the length of the string x. A substring of x is $x_ix_{i+1}\dots x_j$, where $1\leq i,j\leq|x|$. If $i>j$, the substring is the empty string ϵ, where $|\epsilon|=0$. A string distance $d(\cdot,\cdot)$ is a distance over Σ^*. A tokenizer $t(\cdot)$ is a function that maps a string x to a finite multiset of strings $x^t=\{x^{t1},x^{t2},\dots,x^{tm}\}$. We call x^t a tokenized string of x, and we call x^{ti} a token in x^t. Denote the number of tokens in x^t as $\mathcal{T}(x^t)=m$, and the aggregate length of tokens of x^t as $\mathcal{L}(x^t)=\Sigma_i|x^{ti}|$. A simple and commonly used tokenizer splits a string into tokens, i.e., substrings, by using whitespaces as separators. A tokenized-string distance $d^t(\cdot,\cdot)$ is a distance over all tokenized strings.

Note that, given $d^t(\cdot,\cdot)$ and $t(\cdot)$, we can form a string distance by letting $d(x,y)=d^t(t(x),t(y))$. Conversely, we formulate the tokenized-string distances in terms of string distances. Since the tokens of any tokenized string are also strings, string distances can be used to form a tokenized-string distance. More formally, $d^t(t(x),t(y))=d^t(x^t,y^t)=f(d(x^{ti},y^{tj})\forall x^{ti}\in x^t,y^{tj}\in y^t)$, for some function f.

B. Problem Statement

Given two sets of tokenized strings, $R=\{r_1^t,r_2^t,\dots,r_S^t\}$ and $P=\{p_1^t,p_2^t,\dots,p_Q^t\}$, a tokenized-string distance, $d^t(\cdot,\cdot)$, and a threshold, T, the goal is to find all tokenized string pairs $\langle r_s^t,p_q^t\rangle$, s.t. $r_s^t\in R$, $p_q^t\in P$ and $d^t(r_s^t,p_q^t)\leq T$.

The problem can also be expressed in terms of similarity. Given a conversion scheme, λ, from distance to similarity, the goal would be to find all tokenized string pairs whose similarity is at least $\lambda(T)$. Several such schemes are commonly used, e.g., $\lambda(T)=1-T$, $\lambda(T)=1/(1+T)$ or $\lambda(T)=e^{-T}$.

C. String Distances

1) Levenshtein Distance (LD): We borrow the Levenshtein Distance first introduced in [35].

Definition 1: Denote $\langle a\rightarrow b\rangle$ as a character-level edit operation transforming a string a to a string b, where $|a|,|b|$ are 0 or 1. Three character-level edit operations are used.

1) *Insertion* ($|a|=0,|b|=1$)
2) *Deletion* ($|a|=1,|b|=0$)
3) *Substitution* ($|a|=1,|b|=1$)

Given two strings $x,y\in\Sigma^*$, $LD(x,y)$ is the minimum number of character-level edit operations (Insertion, Deletion and Substitution) that transform x to y.

Lemma 1: $LD(\cdot,\cdot)$ is a metric.

LD does not consider the string length or the number of matched characters, which biases the distance to be smaller when comparing relatively short strings. For example, the distance LD("Thomson", "Thompson") $=1$, while LD("Alex", "Alexa") $=1$. While both pairs have the same LD, humans may consider the former pair more similar. This motivates normalizing the Levenshtein Distance.

2) Normalized Levenshtein Distance (NLD): We borrow the Normalized Levenshtein Distance proposed in [37].

Definition 2: For any pair $x,y\in\Sigma^*$, $NLD(x,y)=\frac{2\times LD(x,y)}{|x|+|y|+LD(x,y)}$.

For example, NLD("Thomson", "Thompson") $=\frac{2\times1}{7+8+1}=\frac{1}{8}$, while NLD("Alex", "Alexa") $=\frac{2\times1}{4+5+1}=\frac{1}{5}$. This motivates normalization makes NLD more intuitive than LD.

Lemma 2: $NLD(\cdot,\cdot)\in[0,1]$.

Theorem 1: $NLD(\cdot,\cdot)$ is a metric.

The proof of Lemma 2 is trivial and that of Theorem 1 is in [37]. We introduce the following lemma.

Lemma 3: Assuming $|y|\geq|x|$, the following relationship holds $1-\frac{|x|}{|y|}\leq NLD(x,y)\leq\frac{2}{\frac{|x|}{|y|}+2}$.

D. Tokenized-String Distances

The straightforward way to define a distance between two tokenized strings is to use an existing similarity measure, e.g. Ruzicka, cosine, and Dice [8], to measure the similarity between their token multisets. This proposal is too rigid when considering token edits. A token that belongs to two strings will not be counted as common if it is slightly edited, leading to biasing the distance measures. We propose tokenized-string distances that accommodate token edits[4].

1) Setwise Levenshtein Distance (SLD): We propose two set-level edit operations, and use them, along with character-level edit operations to devise SLD.

Definition 3: Denote $\langle a^t\rightarrow b^t\rangle$ as a set-level edit operation transforming a tokenized string a^t to a tokenized string b^t. Two set-level edit operations are defined.

1) *AddEmptyToken* $\langle a^t\rightarrow a^t\cup\{\epsilon\}\rangle$
2) *RemoveEmptyToken* $\langle a^t\rightarrow a^t\setminus\{\epsilon\}\rangle$

Given two tokenized strings x^t,y^t, $SLD(x^t,y^t)$ is the minimum number of character-level edit operations (Insertion, Deletion and Substitution) performed on tokens with additionally any number of set-level edit operations (AddEmptyToken and RemoveEmptyToken) that transform x^t to y^t.

For example, if $x^t=$ {"chan", "kalan"}, $y^t=$ {"chank", "alan"}, and $z^t=$ {"alan"}. $SLD(x^t,y^t)=2$ by editing "chan" to "chank" and "kalan" to "alan". $SLD(x^t,z^t)=5$ by editing "kalan" to "alan", "chan" to ϵ, and removing the ϵ.

[3]When detecting fraud rings, the joined sets are one and the same, resulting in a self-join, i.e., a pair-wise comparisons of all the tokenized strings.

[4]Other non-metric distances [10], [13], [67] are reviewed in Sec. IV.

Lemma 4: $SLD(\cdot,\cdot)$ is a metric.

2) Normalized Setwise Levenshtein Distance (NSLD): $NSLD$ can be defined in terms of SLD as follows.

Definition 4: Given two tokenized strings x^t, y^t, $NSLD(x^t, y^t) = \frac{2 \times SLD(x^t, y^t)}{\mathcal{L}(x^t)+\mathcal{L}(y^t)+SLD(x^t, y^t)}$.

For example, if $x^t = \{$"chan", "kalan"$\}$ and $y^t = \{$"chank", "alan"$\}$. $NSLD(x^t, y^t) = \frac{2\times 2}{9+9+2} = 0.2$. This motivates normalization makes $NSLD$ more intuitive than SLD.

Lemma 5: $NSLD(\cdot,\cdot) \in [0, 1]$.

Lemma 6: Assuming $\mathcal{L}(y^t) \geq \mathcal{L}(x^t)$, the following relationship holds $1 - \frac{\mathcal{L}(x^t)}{\mathcal{L}(y^t)} \leq NSLD(x^t, y^t) \leq \frac{2}{\frac{\mathcal{L}(x^t)}{\mathcal{L}(y^t)}+2}$.

Theorem 2: $NSLD(\cdot,\cdot)$ is a metric.

By proving $NSLD$ is a metric, it can be leveraged in all flavors of K-nearest-neighbor queries on metric spaces, e.g., [12], [48], [61]. Moreover, the existing algorithms for distributed joining on general metric spaces, e.g., [39], [53], [56], [68], can be leveraged for performing $NSLD$-joins. However, these techniques are not optimized for (tokenized) strings. This motivates developing a specialized framework, Tokenized-String Joiner (TSJ). TSJ leverages existing distributed LD-joining algorithms as building blocks, as discussed in Sec. III.

III. Joining Tokenized Strings

The Tokenized-String Joiner (TSJ) follows a generate-filter-verify paradigm for $NSLD$-joins of tokenized strings. This section discusses how to develop TSJ using MapReduce.

A. The MapReduce Framework

MapReduce [18] has become the *de facto* framework for scalable data processing in shared-nothing clusters, since it offers high scalability and built-in fault tolerance. The computation is expressed in terms of two functions, *map* and *reduce*.

map : $\langle key_1, value_1\rangle \rightarrow [\langle key_2, value_2\rangle]$

reduce : $\langle key_2, [value_2]\rangle \rightarrow [value_3]$

Each record in the input dataset is represented as a tuple $\langle key_1, value_1\rangle$. The input dataset is distributed among the *mappers* that execute the map functionality. Each mapper applies the map function on each input record to produce a list on the form $[\langle key_2, value_2\rangle]$, where [.] represents a list. Then, the *shufflers* group the output of the mappers by the key. Next, each *reducer* is fed a tuple on the form $\langle key_2, [value_2]\rangle$, where $[value_2]$, the reduce_value_list, contains all the $value_2$'s that were output by any mapper with the same key_2 value. Each reducer applies the reduce function on the $\langle key_2, [value_2]\rangle$ tuple to produce a list, $[value_3]$.

B. The Generate-Filter-Verify Paradigm

TSJ follows a generate-filter-verify paradigm for high efficiency. It first generates candidate pairs of tokenized strings. Every generated pair either shares at least one token, or has at least a pair of tokens that are highly similar.

Then, TSJ applies low-cost filters to exclude candidate pairs without loss of correctness. The filters reduce the large number of tokenized string comparisons, and are applied at both the token-level and at the tokenized-string-level. Finally, TSJ calculates the $NSLD$ of the remaining candidate pairs for verification. All the stages are parallelized using MapReduce.

C. Generating Shared-Token Candidate Pairs

To generate all pairs of tokenized strings that share at least one token, each tokenized string in R and P is output with all its tokens. Then, the tokenized strings sharing a token are grouped together for further verification. For efficiency, identifiers of the tokenized strings and the tokens are used.

In MapReduce notation, the same map function processes R and P strings, and is defined as $r_s^t \rightarrow [\langle r_s^{ti}, r_s^t\rangle]$ and $p_q^t \rightarrow [\langle p_q^{tj}, p_q^t\rangle]$. The shufflers group by the key, resulting in each token, z, being associated with all r_s^t and p_q^t tokenized strings containing z. Each reducer receives all the tokenized strings r_s^t and p_q^t containing a shared token, z, and generates the corresponding list of candidates. That is, $\forall r_s^t | z \in r_s^t, \forall p_q^t | z \in p_q^t$, the reduce function is $\langle z, [r_s^t] + [p_q^t]\rangle \rightarrow [\langle r_s^t, p_q^t\rangle]$.

D. Generating Similar-Token Candidate Pairs

TSJ also generates the candidate pairs of tokenized strings that have at least one pair of similar tokens.

Theorem 3: Given two tokenized strings x^t, y^t, and a threshold T, s.t. $NSLD(x^t, y^t) \leq T$. There exists a pair of tokens $\langle x^{ti}, y^{tj}\rangle, x^{ti} \in x^t, y^{tj} \in y^t$ where $NLD(x^{ti}, y^{tj}) \leq T$.

Theorem 3 captures the main insight behind the scalability of TSJ. A pair of tokenized strings where all the possible pairs of their tokens have NLD exceeding the threshold T cannot be a candidate for verification. To the best of our knowledge, $NSLD$ is the first distance measure that guarantees two tokenized strings whose distance is below a threshold T have two tokens where a function of LD of the tokens is below T. This allows for transforming the join from the tokenized-strings domain to the tokens domain that is more manageable. The number of distinct tokens is typically orders of magnitude smaller than that of distinct tokenized strings.

More formally, $NSLD$-joins of tokenized strings can be reduced to the problem of NLD-joins of tokens. To formalize this reduction, we define the token space of a set of tokenized strings to be the set of all the tokens of all of the tokenized strings in the set. Given two sets of tokenized strings, $R = \{r_1^t, r_2^t, \ldots, r_S^t\}$ and $P = \{p_1^t, p_2^t, \ldots, p_Q^t\}$, define their token spaces as $R^t = \{r_s^{ti} | r_s^t \in R, r_s^{ti} \in r_s^t, \forall s, i\}$ and $P^t = \{p_q^{tj} | p_q^t \in P, p_q^{tj} \in p_q^t, \forall q, j\}$, respectively. The first phase of generating $NSLD$-candidates for R and P is generating NLD-candidates for R^t and P^t. For every pair of tokens, $r_s^{ti} \in R^t, p_q^{tj} \in P^t$, such that $NLD(r_s^{ti}, p_q^{tj}) \leq T$, all the tokenized strings in R and P generating r_s^{ti} and p_q^{tj}, respectively, are $NSLD$-join candidates.

To perform the NLD-joins, TSJ employs MassJoin [19], a MapReduce-distributed version of PassJoin [36] that was originally proposed for LD-joins. PassJoin intuition is captured in lemma 7 that is borrowed from [36] .

Lemma 7: Given two strings x, and y, and a threshold U, where $LD(x, y) \leq U$, partitioning y into any $U + 1$ segments results in at least one segment being a substring of x.

PassJoin partitions every token $r_s^{ti} \in R^t$ into its segments, and generates the substrings of every token $p_q^{tj} \in P^t$. For

every matching segment and substring, their generating token pair $\langle r_s^{ti}, p_q^{tj} \rangle$ is a LD-join candidate.

When generating the segments or the substrings of a token, z, the MassJoin mapper outputs z keyed by each of its string chunks (segments or substrings). All tokens sharing the same string chunk are grouped together by the shufflers and are processed by the same reducer. Each reducer outputs all possible candidates pairs of tokens, x and y, from R^t and P^t, respectively. The candidates are de-duplicaated, and the LD-similar pairs of tokens are produced. MassJoin augments the mapper output key by metadata to reduce candidate pairs, and whenever possible, uses unique ids of chunks and tokens. We next explain how to adopt the LD-MassJoin for NLD-joins.

Lemma 8: Given two strings x, and y, and a threshold T, s.t. $NLD(x,y) \leq T$. If $|x| \leq |y|$ then $LD(x,y) \leq \lfloor \frac{2 \times T \times |y|}{2-T} \rfloor$. If $|x| > |y|$, then $LD(x,y) \leq \lfloor \frac{T \times |y|}{1-T} \rfloor$.

Lemma 9: Given two strings x, and y, and a threshold T, s.t. $NLD(x,y) \leq T$. If $|x| \leq |y|$, then $\lceil (1-T) \times |y| \rceil \leq |x|$.

Lemmas 7 and 8 establish a lower bound on the number of segments per token. From lemma 7 any partition scheme is usable. However, an even-partition scheme, where the difference between the shortest and longest generated segments is at most one, reduces the space of string chunks.

Lemmas 8 and 9 establish a condition on the lengths of two tokens to be compared. The *length-condition* $\lceil (1-T) \times |y| \rceil \leq |x| \leq |y|$ has to hold for a pair of tokens, x and y, such that one of which is in R^t and the other is in P^t, and $NLD(x,y) \leq T$.

E. Filtering Candidate Pairs of Tokenized-String

When generating candidate pairs of tokenized strings, relying solely on Theorem 3 results in a large proportion of spurious candidates, especially when the tokenized strings have numerous tokens. TSJ applies two low-cost filters to effectively prune the candidates before computing their $NSLD$.

1) Pruning based on Length: Based on lemma 6, TSJ discards candidate pair of tokenized strings if their aggregate token length ensures their $NSLD$ distance exceeds the threshold, T. The algorithm represents each tokenized string by a unique identifier for efficiency. This identifier is augmented with the length of the tokenized string to prune candidate pairs of identifiers based on the lengths of the tokenized strings.

2) Pruning based on Distance Lower Bound: TSJ can prune candidate pairs by computing a lower bound on the $NSLD$ of any candidate pair by establishing a lower bound on the character-level edit operations for each token of the two tokenized strings. To that end, TSJ augments each tokenized string unique id with a histogram of its token lengths. Given the pair of the token-length histograms of two tokenized strings, and the lengths of the token pairs whose NLD is below T, TSJ can compute lower bounds on SLD and $NSLD$ of the pair by computing a lower bound on the character-level edit operations for all the pairs of tokens, whether matched or unmatched. For the matched tokens, the character-level edit operations are already computed during the candidate generation phase. For the unmatched tokens, TSJ computes a lower bound based on the length histograms, and on Lemma 10.

Lemma 10: Given two strings x, and y, and a threshold T, s.t. $NLD(x,y) > T$. If $|x| \leq |y|$ then $LD(x,y) > \lfloor \frac{T \times |y|}{2-T} \rfloor$. If $|x| > |y|$, then $LD(x,y) > \lfloor \frac{2 \times T \times |y|}{2-T} \rfloor$.

The pruning algorithm and the proof of its correctness will be discussed in an extended version of the paper.

F. The Final Verification

Once the pruned candidates are found, the tokenized-string identifiers are resolved to the tokenized strings, and the final verification is carried by calculating its SLD as below. The $NSLD$ of any pair is trivially computed from its SLD.

SLD Calculation: Given two tokenized strings, x^t, and y^t, let $x^t = \{x^{t1}, x^{t2}, \ldots, x^{tm}\}$ and $y^t = \{y^{t1}, y^{t2}, \ldots, y^{tn}\}$. Calculating $SLD(x^t, y^t)$ entails forming a weighted bigraph $\langle V, E, w \rangle$. Let $k = max(m,n)$. Auxiliary tokenized strings $x^{t\prime} = \{x^{t1}, x^{t2}, \ldots, x^{tk}\}$ and $y^{t\prime} = \{y^{t1}, y^{t2}, \ldots, y^{tk}\}$ are constructed, where extra tokens are empty strings. Let X be a set of nodes that represents tokens in $x^{t\prime}$ and Y be a set of nodes that represents tokens in $y^{t\prime}$. Define $V = X \cup Y$, $E = \{\langle X_i, Y_j \rangle | X_i \in X, Y_j \in Y\}$, and edge weights $w(\langle X_i, Y_j \rangle) = LD(x^{ti}, y^{tj}), \forall i,j$. The minimum weight perfect matching on this weighted bigraph is computed. This is a manifestation of the assignment problem that can be solved using the Hungarian algorithm. The time complexity for constructing the weighted bigraph is $O(\mathcal{L}(x^t) \times \mathcal{L}(y^t))$; and the time complexity of the Hungarian algorithm is $O(max(\mathcal{T}(x^t), \mathcal{T}(y^t))^3)$. So, the overall time complexity of calculating $SLD(x^t, y^t)$ is $O(\mathcal{L}(x^t) \times \mathcal{L}(y^t) + max(\mathcal{T}(x^t), \mathcal{T}(y^t))^3)$.

G. Optimizations and Approximations

The section describes optimizations and approximations.

1) Self-Joins: The motivating application focuses on self-join, i.e., $R = P$. Performing self-join allows for a major optimization, skipping symmetric steps in the candidate generation and the pruning steps. When generating segments, from lemma 8, the case where $|x| \leq |y|$ only needs to be considered, yielding fewer segments. When generating Similar-Token candidate pairs, applying the length-condition does not need to consider the cases where $x \in R^t$, and $y \in P^t$ as well as the cases where $x \in P^t$, and $y \in R^t$, since $P = R$.

2) High-Frequency Tokens: Avoiding high-frequency tokens can enhance the quality of comparison results. In the application of comparing full names described in Sec. I-A, "John" and "Mary" are very common tokens, and may result in uninteresting results. Both the Shared-Token and Similar-Token candidate pair generation processes discard tokens that are shared by more than a given maximum number of tokenized strings, M. Dropping high-frequency tokens achieves better load balancing between the MapReduce workers.

Dropping high-frequency tokens in a scalable way will be discussed in an extended version of the paper.

3) De-Duplicating by Grouping on One String: During the evaluation of the NLD of tokens or the $NSLD$ of the tokenized strings, duplicate pairs arise. To avoid redundant computation, duplicate candidate pairs are discarded using a MapReduce job. There are two possible strategies. The first one, *grouping-on-both-strings* is to use the MapReduce

shuffler to group together instances of the same pair, and make the reducer output this pair exactly once. The second strategy, *grouping-on-one-string*, forms a key-value tuple from each pair, where the key is one string, and the value is the other string. The reducer then de-duplicates the reduce_value_list using a hash set. To balance the load among the reducers, for every pair of strings, τ and υ, τ is used as the key and υ is used as the value if and only if $int(HASH(\tau) < HASH(\upsilon)) = (HASH(\tau) + HASH(\upsilon))\%2$, where $HASH$ is a fingerprint function that hashes τ and υ to an integer. Otherwise, υ becomes the key and τ becomes the value.

4) Exact-Token-Matching Approximation: When the threshold, T, is small, and the tokens are relatively short, a pair of similar tokenized strings tend to share a common token. Then, the candidate-pair generation process can be reduced down to the shared-token strategy. The expensive similar-token strategy can be skipped with minor impact on the recall.

5) Greedy-Token-Aligning Approximation: This approximates the $SLD(x^t, y^t)$ as follows. The edge weights of the token bigraph are computed exactly. However, instead of exactly computing the minimum weight perfect matching using the Hungarian algorithm, a greedy approach is followed. The *greedy-token-aligning* selects the edge with the minimum LD weight, and removes the two corresponding tokens from the bigraph nodes. This is repeated until all the tokens have been matched. This reduces the time complexity to $O(\mathcal{L}(x^t) \times \mathcal{L}(y^t) + \mathcal{T}(x^t) \times \mathcal{T}(y^t) \log(\mathcal{T}(x^t) \times \mathcal{T}(y^t)))$.

IV. Related Work

To the best of our knowledge, this is the first work that introduces a distance measure for tokenized strings that is intuitive, metric, and whose join algorithm is scalable. The previous work on tokenized-string similarity, distributed joins of strings, and distributed fuzzy metric joins are reviewed.

The MGJoin algorithm [51] proposed joining tokenized strings by representing each tokenized string as a bag of tokens. This work proposes a serial algorithm, and a MapReduce extension that employ prefix-filtering [1], [11], [25], [52], [71] and inverted indexes. The technique is very similar to [64], but employs multiple global orders of the tokens. Distributed prefix-filtering-based techniques were shown to be inefficient in [45], as apparent from comparing the speedups against [64] in both [51] and [45]. All these *set-based* techniques handle token shuffles, but do not handle token edits.

In the context of answering K-nearest-neighbor queries for data cleaning, Chaudhuri *et al.* introduced Fuzzy Matching Similarity (FMS) [10]. The user sets penalties for token insertion, deletion, or editing. While being able to handle both token shuffles and edits, FMS had two main drawbacks. First, it is provably not a metric. Second, FMS is sensitive to token order. An approximation, AFMS, was introduced that ignores the token positions. AFMS matches each token in a string to its best matching token in the other string, which may result in multiple tokens from one string matched to the same token in the other string. Unfortunately, both FMS and AFMS are not symmetric, which poses challenges when using them as tokenized-string similarity measures in other applications. Chaudhuri *et al.* proposed a serial FMS-based query algorithm, FuzzyMatch, to identify the closest K tokenized strings given a query, and devised enhancements for indexing, and caching.

Several tokenized-string comparison approaches are contrasted in [13] that focuses on joining names and records by comparing string. Among the contrasted distance measures are the Jaro-based distances [31], [69] that emerged from the statistical communities and deals with names as non-tokenized strings. To overcome the non-tokenization, and the set-based distances drawbacks (no shuffles and no edits, respectively), [13] introduced SoftTfIdf that computes the Jaccard index of two-tokenized strings while considering the popularity of their tokens. SofTfIdf allows for token edits, by allowing tokens to match with up to some threshold on their Jaro-Winkler (JW) distance. However, SoftTfIdf has multiple drawbacks. The work in [13] does not describe an algorithm to compute the distance. To compare two tokenized strings, two thresholds have to be set by the user, $T1$ on the JW distance between tokens and another threshold, $T2$, on the Jaccard similarity of the tokenized strings. Two tokenized strings are considered $T1$-$T2$-similar if their Jaccard similarity exceed $T2$ given that $T1$ is used as the token matching threshold. Setting two unrelated thresholds impairs the tuning of the join. SoftTfIdf is non-metric, since JW violates the triangle inequality.

The work in [67] is a generalization and an improvement over that in [13]. In addition to Jaccard, the work in [67] adapts more set-based similarity measures [8], e.g., Dice, cosine, to tokenized-string joins. Like SoftTfIdf [13], to compare two tokenized strings, Wang *et al.* require a threshold, $T1$ on token similarity and another threshold, $T2$, on tokenized-string similarity. The algorithm matches two tokens from the two token sets only if their token similarity exceed $T1$. The sum of the similarities of the matched tokens are then used to compute the customized set-based similarity of the two tokenized strings. Like [10], [13], the similarity measure in [67] depends on two totally unrelated thresholds, which impairs the tuning of the join. Like [10], [13], the proposed tokenized-string measures are provably non-metric.

Wang *et al.* devised an effective candidate string-pair pruning technique that works for the customized similarity measures in [67], and is based on prefix-filtering. The custom pruning proposed in [67] imposes limitations on adopting new string-joining algorithms. Moreover, the algorithm is serial, i.e., cannot scale out to multiple machines for scalability.

When compared to the existing tokenized-string distance measures, the proposed $NSLD$ is a metric, and naturally uses the tokenized-string threshold, T, to prune non-similar tokens (Theorem 3). The TSJ framework uses a state-of-the-art distributed string-joining algorithms to perform the joins, and can hence scale out virtually to any input dataset.

Several string-join algorithms have been proposed, as recently surveyed in [73], and their performance was compared in [9], [13], [26]. In our implementation, we used MassJoin [19] for finding similar tokens. MassJoin is a distributed version of Pass-Join [36], which employs a filter-and-verification

framework. In the filter step, Pass-Join generates signatures for strings, such that two similar strings are guaranteed to share at least one signature. This way, Pass-Join prunes pairs of strings that are guaranteed to be dissimilar. The remaining candidate pairs are verified in the verification step. Pass-Join was also generalized such that two similar strings are guaranteed to share K signatures, yielding the PassJoinK algorithm [38]. The PassJoinK was parallelized, yielding the PassJoinKMR and PassJoinKMRS algorithms [38]. As noted in [73], the MassJoin avoids shortcomings of its competitors by frugally generating candidates, and employing light-weight filters.

Distributed joining in general metric spaces has been the focus of [39], [53], [56], [68], most of which rediscover or borrow ideas from the work in [30]. These efforts (recursively) partition the data records into sub-partitions such that similar records belong to the same partitions, or neighboring partitions in the metric space. These techniques compare only two partitions if their candidate records may yield a joined pair. In theory, these techniques can be employed to compare tokenized strings as well as tokens, since both $NSLD$ and NLD are metrics. However, in preliminary experiments, none of these techniques ran to completion within reasonable time when applied to $NSLD$-joins as detailed in Sec. V.

V. Evaluation

In this section, we report the results of several experiments that evaluate the scalability and the accuracy of TSJ, and the impact of the proposed optimizations and approximations.

The motivation for this work is detecting rings of accounts using their tokenized strings signals. The experiments were run on the names on Google accounts from a specific region. The number of tokenized strings used is 44,382,766. The names were tokenized using whitespaces and punctuation characters. The tokenized strings were self-joined resulting in 1.9670×10^{15} possible pairs.

The default parameters used for the evaluation runs are as follows. The MapReduce jobs are run on 1,000 machines (1,000 Mappers and 1,000 Reducers) running Ubuntu Linux. Every machine is allowed only 1G of memory, 5G of disk, and 0.5 cpu. The default thresholds on the pairwise $NSLD$ (T), and on popular tokens occurring with multiple tokenized strings (M) assume the values 0.1 and 1,000, respectively[5].

A. Scalability and Speedup

TSJ was run while varying the number of machines from 100 to 1,000. Both options for deduping the candidate pairs of strings (grouping-on-one-string and grouping-on-both-strings) were used, and the runtimes are plotted in Fig. 1. Both deduping options scaled out well. Both achieved a speedup of 3.8 as the number of machines increased by 10 folds.

[5]These parameters may differ across geo-locations depending on the names popularity. Typically, for each major geo-location, a gradient descent search is performed to set these parameters. At each gradient descent evaluation, a sample of the clusters is evaluated by the operations team of Google, and the rates of true positives and the false positives are computed. The values of 0.1 and 1,000 constitute a reasonable starting point for the search. For this dataset, setting M to 1,000 discarded roughly 1% of the tokens.

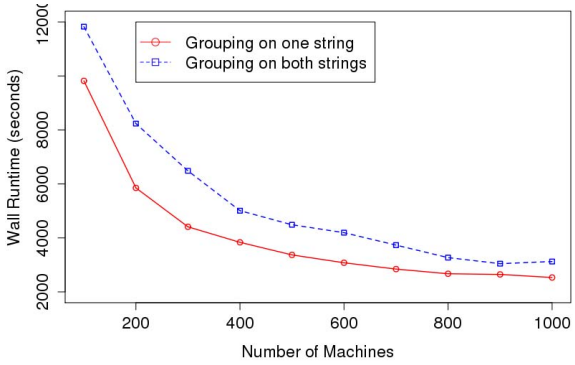

Fig. 1. Comparing the runtime of Tokenized-String Joiner (TSJ) while varying the MapReduce machines and the Deduping algorithm.

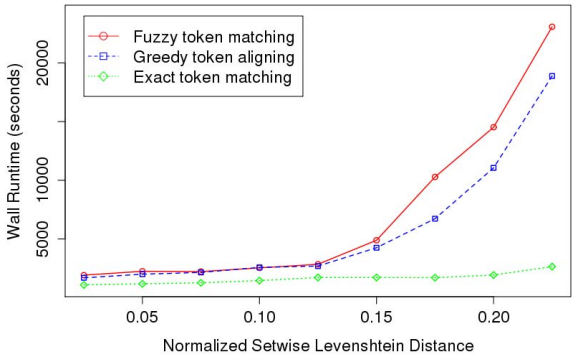

Fig. 2. Comparing the runtime of Tokenized-String Joiner (TSJ) while varying NSLD and the token matching and aligning algorithms.

Consistently, grouping-on-one-string was clearly faster than grouping-on-both-strings, achieving a speedup between 13% and 32%. This can be attributed to the overhead of instantiating MapReduce workers. The grouping-on-one-string mechanism instantiates a worker for each string, whose work is verifying each of its potentially similar strings. Meanwhile, grouping-on-both-strings instantiates a worker for each candidate pair of strings, whose work is verifying only one pair of strings.

Notice that grouping-on-both-strings achieves better load balancing. In case there exists a small set of strings, each of which is potentially similar to numerous strings, all these candidate pairs would be spread out among multiple workers, which better distributes the load than grouping-on-one-string.

B. Joining by Approximate Matching

The effect of the approximations in Sec. III-G, greedy-token-aligning and exact-token-matching, on runtime and accuracy are reported.

1) Impact of Approximations on Speedup: All the experiments below are run using grouping-on-one-string. While fuzzy-token-matching produces the correct pairs of similar tokenized strings, it has the longest runtime.

Fig. 2 shows the runtime while varying T from 0.025 to 0.225. The mean runtime saving of greedy-token-aligning over fuzzy-token-matching is 13%, and is more pronounced as T increases. The mean runtime saving of exact-token-matching over fuzzy-token-matching is 60%. Moreover, the runtime of exact-token-matching increases only slightly as T increases.

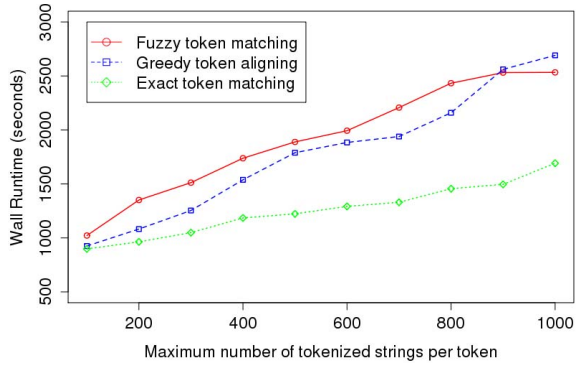

Fig. 3. Comparing the runtime of Tokenized-String Joiner (TSJ) while varying max-frequency (M) and the token matching and aligning algorithms.

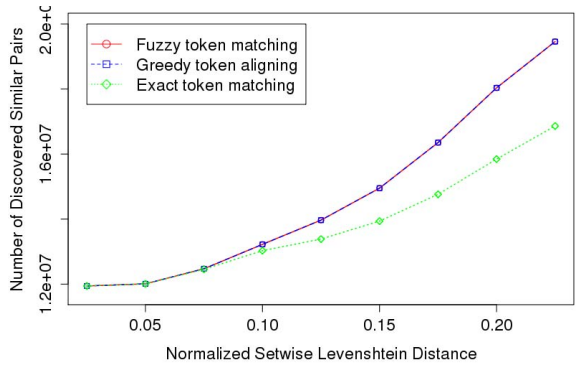

Fig. 4. Comparing the number of pairs of Tokenized-String Joiner (TSJ) while varying NSLD and the token matching and aligning algorithms.

Fig. 3 shows the runtime while varying M from 100 to 1,000. The mean runtime saving of greedy-token-aligning over fuzzy-token-matching is 9%, while that of exact-token-matching is 33%. The runtime savings of both approximation schemes were fairly stable across the values of M.

2) Impact of Approximations on Accuracy: The proposed approximations make TSJ err on the false negative side, guaranteeing the precision (the percentage of the discovered pairs that are truly similar) to be always 1.0. However, the recall (the ratio between the number of the discovered pairs to the number of pairs discovered by fuzzy-token-matching) is not necessarily as perfect. To assess the impact of the proposed approximations on the recall, the number of pairs of tokenized strings that were found similar were reported while varying the two main parameters, T and M. The numbers of similar pairs are plotted in Fig. 4 and Fig. 5, respectively.

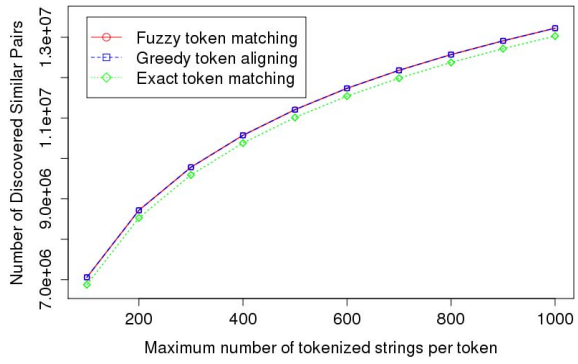

Fig. 5. Comparing the number of pairs of Tokenized-String Joiner (TSJ) while varying max-frequency (M) and the token matching and aligning algorithms.

Fig. 4 shows the number of discovered pairs while varying T from 0.025 to 0.225. The recall of greedy-token-aligning (exact-token-matching) was 1.0 when T was 0.025 and decreased to 0.99993 (0.86655) as T reached 0.225.

Fig. 5 shows the number of discovered pairs while varying M from 100 to 1,000. For all values of M, the recall of greedy-token-aligning was stable around 0.999999, and between 0.974 and 0.985 for exact-token-matching.

In the space of possible (T, M) thresholds, some deductions could be made around the reasonable point of $(0.1, 1{,}000)$ from Fig. 4 and Fig. 5. First, the number of similar pairs of tokenized strings increases more aggressively by increasing T than by increasing M. Second, increasing T has more impact on the recall of the approximations proposed in Sec. III-G than M. Below we analyze why this is the case for each of the two approximations in turn.

As T increases, there is more room for fuzzy-token-matching to behave differently from greedy-token-aligning. When aligning the tokens of any pair of tokenized strings, at least a pair of aligned tokens has its NLD smaller than T (Theorem 3). Hence, larger T translates to more ways to align the tokens of the two tokenized strings, making greedy-token-aligning finding the optimal alignment less likely.

On the contrary, an increase in M does not highlight the difference between fuzzy-token-matching and greedy-token-aligning. An increase in M merely increases the number of tokens considered for generating candidate pairs of tokenized strings, which does not impact the token aligning.

The increase in T leaves room for fuzzy-token-matching to behave differently from exact-token-matching in generating the candidate pairs of tokenized strings. For any pair of tokenized strings to have their $NSLD$ below the T, at least two tokens have to exist, one from each tokenized string, such that their NLD is below T (Theorem 3). For small T and relatively short name strings, small NLD increases the chance of an exact match between these two tokens, which makes fuzzy-token-matching degenerate to exact-token-matching.

On the other hand, the increase in M highlights the difference between fuzzy-token-matching and greedy-token-aligning to a lesser degree. Any similar pair of tokenized strings either share a token, or have a pair of tokens that can be fuzzily matched together. In the first case, exact-token-matching behave exactly like fuzzy-token-matching. In the second case, only fuzzy-token-matching discovers such candidate pairs. Since as M increases, the tokens that become considered for discovering candidate pairs are popular, the first case becomes more prevalent.

C. Lessons from Advertising Fraud Rings

The TSJ framework was very effective in catching fraud rings. Using multiple string signals, TSJ allowed for discovering several fraud rings that were not detectable previously. Based on the evaluation above that was done on real full

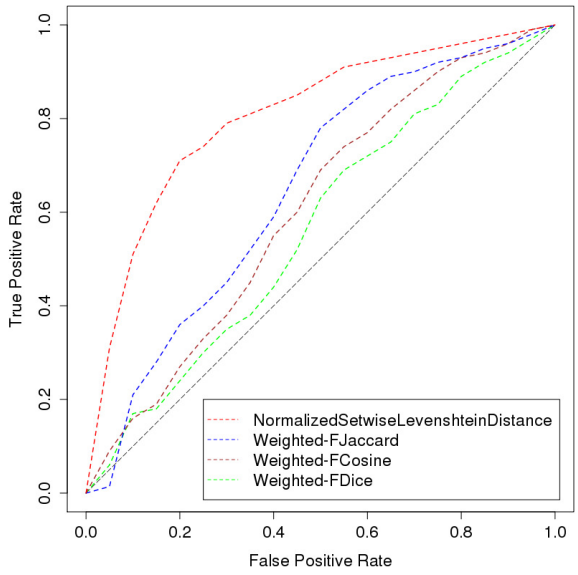

Fig. 6. The ROC curves of NSLD, wighted FJaccard, weighted FCosine, and weighted FDice when predicting fraudulent accounts based on the distance between the old and new names on an account.

names, we strongly recommend using greedy-token-aligning for all values of T, and M. This results in almost no loss in recall, and notable enhancement in the runtime. We also recommend implementing grouping-on-one-string since it resulted in improving the runtime for all the experiments.

Using exact-token-matching when T is set to a small or a moderate value (between 0.025 and 0.1) results in very minor loss in recall, with very significant improvement in the runtime. However, It is worth noting that the exact-token-matching approximation caught mainly the unsophisticated attacks that had the least monetary impact. The more sophisticated attacks were caught only by the fuzzy-token-matching algorithm. Hence, we recommend this approximation only for data integration and cleaning where missing some similar records does not have a significant financial impact, and the computational resources are scarce.

D. Comparing the Accuracy of the Distance Measures

The distances produced by $NSLD$ and the state-of-the-art distance measures are compared. The comparison was with the weighted versions of the set-based fuzzy similarity measures, $FJaccard$, $FCosine$, and $FDice$ [67]. For all the set-based fuzzy measures, the distance is taken as $1 - similarity$.

We could not evaluate the quality of the set-based fuzzy similarity measures on our dataset, since the join algorithm proposed in [67] is serial and does not scale to large-scale datasets. However, we assessed how $NSLD$ and the set-based fuzzy measures can be used as proxies to assess if an account is fraudulent or not. We have selected a sample of 10,000 accounts that have their names changed. Half the sample were known legitimate accounts and the other half were known fraudulent accounts. The name changes on the legitimate accounts happen in rare cases, such as legal name changes, or name abbreviation, e.g., from "`William`" to "`Bill`". On the other hand, name changes on fraudulent accounts are usually very drastic, since, attackers who specialize in account creation are not those who specialize in account exploitation [60]. The account-creation attacker typically chooses a random name. When the credentials are sold to the attacker, the account name is drastically changed. The $NSLD$ and the set-based fuzzy measures were used to measure the distance between the old and the new account names. The ROC curves are in Fig. 6.

Assuming the correlation between the magnitude of the name change and the likelihood of fraud, $NSLD$ is superior to all these set-based fuzzy measures when quantifying the distances between names on accounts. The existing tokenized-string measures assume the edits are mostly non-malicious and are either manual typos or OCR errors. This assumption does not apply when an adversary strives to game the measures.

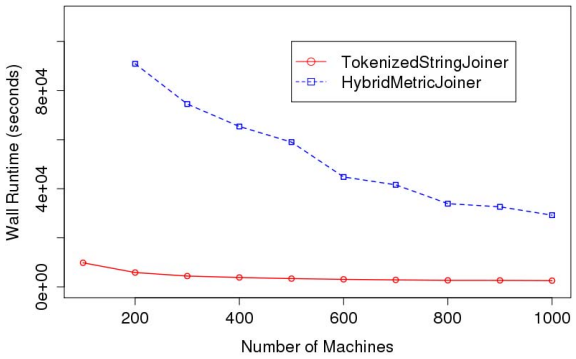

Fig. 7. Comparing the runtime of Tokenized-String Joiner (TSJ) and the Hybrid Metric Joiner (HMJ) while varying the MapReduce machines.

E. Comparing TSJ with the State of the Art Join Algorithms

TSJ is the first distributed similarity-join algorithm for tokenized strings. However, the existing distributed metric-space join algorithms lend themselves to a comparison with TSJ, since the proposed distance, $NSLD$, is a metric.

We compared TSJ to a state-of-the-art metric-space join algorithm. The Hybrid Metric Joiner (HMJ) is an in-house-built algorithm that is hybrid of the most scalable and efficient algorithms [53], [68] proposed for metric-space joins and reviewed in Sec. IV. As proposed in [53], the tokenized strings are dissected into partitions among centroids by dissecting the space among the centroids using Voronoi hyperplanes, and are assigned to neighboring partitions using the general filter in [53]. The symmetry of the distance metric was exploited to reduce comparing tokenized strings from neighboring partitions, as proposed in [68]. Each partition is recursively repartitioned either using sub-centroids as proposed in [68], or using a 2-dimensional grid as proposed in [53], depending on how the tokenized strings are scattered within the partition. Finally, the computation is made more efficient by exploiting the triangle inequality to output cliques and bicliques of tokenized strings as described in [68]. TSJ and HMJ were run while varying the number of machines from 100 to 1,000, and the runtimes are plotted in Fig. 7. HMJ did not finish on 100 machines in a reasonable amount of time. For all the other configs, TJS was 12 to 15 times faster than HMJ.

The metric-space techniques work best when the data is fairly distributed in the space. However, when visualizing the tokenized strings as multi-dimensional points in a metric space, they form a large number of fairly dense clusters, due to sharing tokens. This results in load imbalance between the workers. TJS on the other hand transforms the join problem to the token domain, performs a join using a very specialized and scalable string-join algorithm, and uses the results to generate candidate pairs of tokenized strings. Since most of the work happens in the token domain, the poorly distributed tokenized strings do not impact the performance.

VI. Conclusion

Existing tokenized-string distance measures are sensitive to the order of the tokens, asymmetric, or are hard to tune since they require setting multiple independent thresholds. Moreover, their join algorithms are serial and hence unscalable.

In this paper, we motivated and introduced a novel distance metric between tokenized strings, Normalized Setwise Levenshtein Distance ($NSLD$). To the best of our knowledge, $NSLD$ is the first metric distance for tokenized strings. Moreover, $NSLD$ is the only distance measure that guarantees carrying the distance threshold from the tokenized strings to their tokens. This allows for transforming the join from the tokenized-string domain to the tokens domain. The tokens domain is more manageable, since the number of distinct tokens is typically orders of magnitude smaller than that of distinct tokenized strings, and the literature of string-join algorithm is richer. Based on this property, we propose a very specialized and scalable framework, Tokenized-String Joiner (TSJ), which adopts existing string-join algorithms as building blocks to perform $NSLD$-based joins. The scalability of the proposed framework, and the effectiveness and accuracy of the proposed approximations are established by our evaluation on tens of millions of names on Google accounts. We demonstrated the superiority of $NSLD$ over the state-of-the-art weighted set-based fuzzy similarity measures in terms of accuracy, and the superiority of the tokenized-string-specific TSJ over the metric-spaces joins in terms of scalability and efficiency.

References

[1] R. Bayardo, Y. Ma, and R. Srikant. Scaling Up All Pairs Similarity Search. In *Proceedings of the 16th WWW International Conference on World Wide Web*, pages 131–140, 2007.

[2] Y. Boshmaf, I. Muslukhov, K. Beznosov, and M. Ripeanu. The Socialbot Network: When Bots Socialize for Fame and Money. In *Proceedings of the 27th Annual Computer Security Applications Conference*, pages 93–102, 2011.

[3] R. Brandom. A New $10 Million-a-Month Fraud Ring Preys on Online Advertisers, 2014. http://www.theverge.com/2014/4/18/5628028/10-million-a-month-fraud-ring-preys-on-online-advertisers.

[4] I. A. Bureau. IAB Internet Advertising Revenue Report 2017, Full Year Results, 2018. https://www.iab.com/wp-content/uploads/2018/05/IAB-2017-Full-Year-Internet-Advertising-Revenue-Report.REV2_.pdf.

[5] E. Bursztein, B. Benko, D. Margolis, T. Pietraszek, A. Archer, A. Aquino, A. Pitsillidis, and S. Savage. Handcrafted Fraud and Extortion: Manual Account Hijacking in the Wild. In *Proceedings of the IMC Internet Measurement Conference*, pages 347–358, 2014.

[6] Q. Cao, M. Sirivianos, X. Yang, and T. Pregueiro. Aiding the Detection of Fake Accounts in Large Scale Social Online Services. In *Proceedings of the 9th USENIX NSDI Symposium on Networked Systems Design and Implementation*, pages 197–210, 2012.

[7] Q. Cao, X. Yang, J. Yu, and C. Palow. Uncovering Large Groups of Active Malicious Accounts in Online Social Networks. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, pages 477–488, 2014.

[8] S.-H. Cha. Comprehensive Survey on Distance/Similarity Measures between Probability Density Functions. *International Journal of Mathematical Models and Methods in Applied Sciences*, 1(4):300–307, 2007.

[9] A. Chandel, O. Hassanzadeh, N. Koudas, M. Sadoghi, and D. Srivastava. Benchmarking Declarative Approximate Selection Predicates. In *Proceedings of the 27th ACM SIGMOD international conference on Management of data*, pages 353–364, 2007.

[10] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani. Robust and Efficient Fuzzy Match for Online Data Cleaning. In *Proceedings of the 23rd ACM SIGMOD international conference on Management of data*, pages 313–324, 2003.

[11] S. Chaudhuri, V. Ganti, and R. Kaushik. A Primitive Operator for Similarity Joins in Data Cleaning. In *Proceedings of the 22nd IEEE International Conference on Data Engineering*, page 5, 2006.

[12] L. Chen, Y. Gao, G. Chen, and H. Zhang. Metric All-k-Nearest-Neighbor Search. *IEEE Transactions on Knowledge and Data Engineering*, 28(1):98–112, 2016.

[13] W. Cohen, P. Ravikumar, and S. Fienberg. A Comparison of String Metrics for Matching Names and Records. In *Kdd workshop on data cleaning and object consolidation*, volume 3, pages 73–78, 2003.

[14] K. Conger. Amazon Sues Sellers for Buying Fake Reviews, 2016. https://techcrunch.com/2016/06/01/amazon-sues-sellers-for-buying-fake-reviews/.

[15] V. Dave, S. Guha, and Y. Zhang. Measuring and Fingerprinting Click-spam in Ad Networks. In *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 175–186, 2012.

[16] V. Dave, S. Guha, and Y. Zhang. ViceROI: Catching Click-Spam in Search Ad Networks. In *Proceedings of the ACM CCS Conference on Computer and Communications Security*, pages 765–776, 2013.

[17] E. De Cristofaro, A. Friedman, G. Jourjon, M. Kaafar, and Z. Shafiq. Paying for Likes? Understanding Facebook Like Fraud using Honeypots. In *Proceedings of the IMC Conference on Internet Measurement Conference*, pages 129–136, 2014.

[18] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *Proceedings of the 6th OSDI Symposium on Opearting Systems Design & Implementation*, pages 10–10, 2004.

[19] D. Deng, G. Li, S. Hao, J. Wang, and J. Feng. MassJoin: A MapReduce-based Method for Scalable String Similarity Joins. In *IEEE 30th ICDE International Conference on Data Engineering*, pages 340–351, 2014.

[20] M. Egele, G. Stringhini, C. Kruegel, and G. Vigna. COMPA: Detecting Compromised Accounts on Social Networks. In *20th NDSS Annual Network and Distributed System Security Symposium*, 2013.

[21] D. M. Freeman, S. Jain, M. Dürmuth, B. Biggio, and G. Giacinto. Who Are You? A Statistical Approach to Measuring User Authenticity. In *23rd Annual Network & Distributed System Security Symposium (NDSS)*, 2016.

[22] V. Ganti and A. Sarma. *Data Cleaning: A Practical Perspective*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2013.

[23] J. Gatto. LinkedIn Has Beef With Data Scraping Bots, 2016. http://www.lawofthelevel.com/2016/08/articles/privacy/linkedin-has-beef-with-data-scraping-bots/.

[24] Google, Inc. If You Want to Apply for More Than One AdSense Account, 2018. https://support.google.com/adsense/answer/9729.

[25] L. Gravano, P. Ipeirotis, H. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. Approximate String Joins in a Database (Almost) for Free. In *Proceedings of the 27th VLDB International Conference on Very Large Data Bases*, volume 1, pages 491–500, 2001.

[26] O. Hassanzadeh, M. Sadoghi, and R. Miller. Accuracy of Approximate String Joins Using Grams. In *QDB*, pages 11–18, 2007.

[27] K. Hoffman, D. Zage, and C. Nita-Rotaru. A survey of Attack and Defense Techniques for Reputation Systems. *ACM Computing Surveys (CSUR)*, 42(1):1, 2009.

[28] M. Ikram, L. Onwuzurike, S. Farooqi, E. De Cristofaro, A. Friedman, G. Jourjon, M. Kaafar, and M. Shafiq. Measuring, Characterizing, and Detecting Facebook Like Farms. *ACM Transactions on Privacy and Security (TOPS)*, 20(4):13, 2017.

[29] N. Ingraham. Twitter's Troll Problem Likely Killed Disney's Bid, 2016. https://www.engadget.com/2016/10/18/disney-twitter-purchase-troll-problem/.

[30] E. Jacox and H. Samet. Metric Space Similarity Joins. *ACM Transactions on Database Systems (TODS)*, 33(2):7, 2008.

[31] M. Jaro. Probabilistic Linkage of Large Public Health Data Files. *Statistics in Medicine*, 14(5-7):491–498, 1995.

[32] L. Jin, H. Takabi, and J. Joshi. Towards Active Detection of Identity Clone Attacks on Online Social Networks. In *Proceedings of the 1st Conference on Data and Application Security and Privacy*, pages 27–38, 2011.

[33] R. Kailath. Some Amazon Reviews Are Too Good To Be Believed. They're Paid For, 2018. https://www.npr.org/2018/07/30/629800775/some-amazon-reviews-are-too-good-to-be-believed-theyre-paid-for.

[34] A. Kakhki, C. Kliman-Silver, and A. Mislove. Iolaus: Securing Online Content Rating Systems. In *Proceedings of the 22nd WWW International Conference on World Wide Web*, pages 919–930, 2013.

[35] V. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions, and Reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.

[36] G. Li, D. Deng, J. Wang, and J. Feng. Pass-Join: A Partition-based Method for Similarity Joins. *Proceedings of the VLDB Endowment*, 5(3):253–264, 2011.

[37] Y. Li and B. Liu. A Normalized Levenshtein Distance Metric. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(6):1091–1095, 2007.

[38] C. Lin, H. Yu, W. Weng, and X. He. Large-Scale Similarity Join with Edit-Distance Constraints. In *DSFAA International Conference on Database Systems for Advanced Applications*, pages 328–342, 2014.

[39] W. Liu, Y. Shen, and P. Wang. An Efficient MapReduce Algorithm for Similarity Join in Metric Spaces. *The Journal of Supercomputing*, 72(3):1179–1200, 2016.

[40] D. McCoy, H. Dharmdasani, C. Kreibich, G. Voelker, and S. Savage. Priceless: The Role of Payments in Abuse-advertised Goods. In *Proceedings of the ACM CCS Conference on Computer and Communications Security*, pages 845–856, 2012.

[41] A. Metwally, D. Agrawal, and A. El Abbadi. Duplicate Detection in Click Streams. In *Proceedings of the 14th WWW International Conference on World Wide Web*, pages 12–21, 2005.

[42] A. Metwally, D. Agrawal, and A. El Abbadi. DETECTIVES: DETEcting Coalition hiT Inflation attacks in adVertising nEtworks Streams. In *Proceedings of the 16th WWW International Conference on World Wide Web*, pages 241–250, 2007.

[43] A. Metwally, D. Agrawal, A. El Abbadi, and Q. Zheng. On Hit Inflation Techniques and Detection in Streams of Web Advertising Networks. In *27th ICDCS International Conference on Distributed Computing Systems*, pages 52–52, 2007.

[44] A. Metwally, F. Emekçi, D. Agrawal, and A. El Abbadi. SLEUTH: Single-pubLisher attack dEtection Using correlaTion Hunting. *Proceedings of the VLDB Endowment*, 1(2):1217–1228, 2008.

[45] A. Metwally and C. Faloutsos. V-SMART-Join: A Scalable MapReduce Framework for All-Pair Similarity Joins of Multisets and Vectors. *Proceedings of the VLDB Endowment*, 5(8):704–715, 2012.

[46] A. Metwally and M. Paduano. Estimating the Number of Users behind IP Addresses for Combating Abusive Traffic. *ACM SIGKDD 17th International Conference on Knowledge Discovery and Data Mining*, pages 249–257, 2011.

[47] A. Metwally, J.-Y. Pan, M. Doan, and C. Faloutsos. Scalable Community Discovery from Multi-Faceted Graphs. In *Proceedings' of the 3rd IEEE BigData International Conference on Big Data*, pages 1053–1062, 2015.

[48] M. Muja and D. Lowe. Scalable Nearest Neighbor Algorithms for High Dimensional Data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11):2227–2240, 2014.

[49] S. Rahman, T.-K. Huang, H. Madhyastha, and M. Faloutsos. Detecting Malicious Facebook Applications. *IEEE/ACM Transactions on Networking*, 24(2):773–787, 2016.

[50] A. Romano. Academic Journals are Facing a Battle to Weed out Fake Peer Reviews, 2015. http://www.dailydot.com/parsec/academic-journals-retraction-peer-review-scam/.

[51] C. Rong, W. Lu, X. Wang, X. Du, Y. Chen, and A. Tung. Efficient and Scalable Processing of String Similarity Join. *IEEE Transactions on Knowledge and Data Engineering*, 25(10):2217–2230, 2013.

[52] S. Sarawagi and A. Kirpal. Efficient Set Joins on Similarity Predicates. In *Proceedings of the 24th ACM SIGMOD International Conference on Management of Data*, pages 743–754, 2004.

[53] A. D. Sarma, Y. He, and S. Chaudhuri. ClusterJoin: A Similarity Joins Framework using Map-Reduce. *Proceedings of the VLDB Endowment*, 7(12):1059–1070, 2014.

[54] E. Schonfeld. The Evolution Of Click Fraud: Massive Chinese Operation DormRing1 Uncovered, 2009. https://techcrunch.com/2009/10/08/the-evolution-of-click-fraud-massive-chinese-operation-dormring1-uncovered/.

[55] B. Schwartz. Google's Matt Cutts: That Scraper Isn't Hurting Your Mom's Site, 2013. https://www.seroundtable.com/google-scraper-success-17558.html.

[56] Y. Silva, J. Reed, and L. Tsosie. MapReduce-based Similarity Join for Metric Spaces. In *Proceedings of the 1st International Workshop on Cloud Intelligence*, page 3, 2012.

[57] G. Sloane. Fraud Alert: Millions of Video Views Faked in Sophisticated New Bot Scam, 2014. http://www.adweek.com/news/technology/fraud-alert-millions-video-views-faked-sophisticated-new-bot-scam-156883.

[58] F. Soldo and A. Metwally. Traffic Anomaly Detection Based on the IP Size Distribution. In *Proceedings of the 31st IEEE INFOCOM International Conference on Computer Communications*, pages 2005–2013, 2012.

[59] O. Stitelman, C. Perlich, B. Dalessandro, R. Hook, T. Raeder, and F. Provost. Using Co-visitation Networks for Detecting Large Scale Online Display Advertising Exchange Fraud. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1240–1248, 2013.

[60] K. Thomas, D. McCoy, C. Grier, A. Kolcz, and V. Paxson. Trafficking Fraudulent Accounts: The Role of the Underground Market in Twitter Spam and Abuse. In *Proceedings of the 15th conference on USENIX Security Symposium*, pages 195–210, 2013.

[61] E. Tiakas, G. Valkanas, A. Papadopoulos, Y. Manolopoulos, and D. Gunopulos. Processing Top-k Dominating Queries in Metric Spaces. *ACM Transactions on Database Systems (TODS)*, 40(4):23, 2016.

[62] N. Tran, J. Li, L. Subramanian, and S. Chow. Optimal Sybil-Resilient Node Admission Control. In *Proceedings of the IEEE INFOCOM International Conference on Computer Communications*, pages 3218–3226, 2011.

[63] T. Vega. Facebook Says It Failed to Bar Posts With Hate Speech, 2013. http://www.nytimes.com/2013/05/29/business/media/facebook-says-it-failed-to-stop-misogynous-pages.html.

[64] R. Vernica, M. Carey, and C. Li. Efficient Parallel Set-Similarity Joins using MapReduce. In *Proceedings of the 30th ACM SIGMOD International Conference on Management of data*, pages 495–506, 2010.

[65] A. Wang. Don't Follow Me: Spam Detection in Twitter. In *Proceedings of the SECRYPT International Conference on Security and Cryptography*, pages 1–10, 2010.

[66] G. Wang, T. Konolige, C. Wilson, X. Wang, H. Zheng, and B. Zhao. You are How You Click: Clickstream Analysis for Sybil Detection. In *Proceedings of the 22nd USENIX Security Symposium*, pages 241–256, 2013.

[67] J. Wang, G. Li, and J. Feng. Extending String Similarity Join to Tolerant Fuzzy Token Matching. *ACM Transactions on Database Systems (TODS)*, 39(1):7, 2014.

[68] Y. Wang, A. Metwally, and S. Parthasarathy. MR-MAPSS: All-Pair Similarity Search in Metric Spaces Using MapReduce. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 829–837, 2013.

[69] W. Winkler. The state of Record Linkage and Current Research Problems. In *Statistical Research Division, US Census Bureau*, 1999.

[70] C. Xiao, D. Freeman, and T. Hwa. Detecting Clusters of Fake Accounts in Online Social Networks. In *Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security*, pages 91–101, 2015.

[71] C. Xiao, W. Wang, X. Lin, J. Yu, and G. Wang. Efficient Similarity Joins for Near-Duplicate Detection. *ACM Transactions on Database Systems (TODS)*, 36(3):15, 2011.

[72] Z. Yang, C. Wilson, X. Wang, T. Gao, B. Zhao, and Y. Dai. Uncovering Social Network Sybils in the Wild. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 8(1):2, 2014.

[73] M. Yu, G. Li, D. Deng, and J. Feng. String Similarity Search and Join: A Survey. *Frontiers of Computer Science*, 10(3):399–417, 2016.

[74] L. Zhang, S. Jiang, J. Zhang, and W.-K. Ng. Robustness of Trust Models and Combinations for Handling Unfair Ratings. In *IFIP International Conference on Trust Management*, pages 36–51, 2012.

[75] Y. Zhao, Y. Xie, F. Yu, Q. Ke, Y. Yu, Y. Chen, and E. Gillum. BotGraph: Large Scale Spamming Botnet Detection. In *Proceedings of the 6th USENIX NSDI Symposium on Networked Systems Design and Implementation*, volume 9, pages 321–334, 2009.

APPENDIX

A. Proof of Lemma 3

Since $|y| \geq |x|$, and from Def. 2, $NLD(x,y) = \frac{2}{\frac{|x|+|y|}{LD(x,y)}+1}$, then $|y| - |x| \leq LD(x,y) \leq |y|$.

B. Proof of Lemma 4

Trivially, $SLD(x^t, x^t) = 0$. Moreover, $SLD(x^t, y^t) = SLD(y^t, x^t) \forall x^t, y^t$. To prove the Triangle Inequality, let $x^t = \{x^{t1}, x^{t2}, \ldots, x^{tm}\}$, $y^t = \{y^{t1}, y^{t2}, \ldots, y^{tn}\}$, and $z^t = \{z^{t1}, z^{t2}, \ldots, z^{to}\}$. Let $k = max\{m,n,o\}$. Construct three auxiliary tokenized strings $x^{t\prime} = \{x^{t1}, x^{t2}, \ldots, x^{tk}\}$, $y^{t\prime} = \{y^{t1}, y^{t2}, \ldots, y^{tk}\}$, and $z^{t\prime} = \{z^{t1}, z^{t2}, \ldots, z^{tk}\}$, where extra tokens are empty strings. Clearly, $SLD(x^t, y^t) = SLD(x^{t\prime}, y^{t\prime})$, $SLD(y^t, z^t) = SLD(y^{t\prime}, z^{t\prime})$, and $SLD(x^t, z^t) = SLD(x^{t\prime}, z^{t\prime})$. There is a transforming path for each token tuple $x^{tx_i} \rightarrow y^{ty_i} \rightarrow z^{tz_i}, \forall i \in [1,k]$. From the Triangle Inequality of $LD(\cdot,\cdot)$, it follows that $LD(x^{tx_i}, y^{ty_i}) + LD(y^{ty_i}, z^{tz_i}) \geq LD(x^{tx_i}, z^{tz_i})$. Summing on all k tokens, $\sum_i^k LD(x^{tx_i}, y^{ty_i}) + \sum_i^k LD(y^{ty_i}, z^{tz_i}) \geq \sum_i^k LD(x^{tx_i}, z^{tz_i})$. Hence, $SLD(x^t, y^t) + SLD(y^t, z^t) \geq SLD(x^t, z^t)$.

C. Proof of Lemma 5

Trivially, $NSLD(x^t, x^t) = 0$. In the other extreme, let x^t only be empty. $\mathcal{L}(x^t) = 0$. From Def. 3, $SLD(x^t, y^t) = \mathcal{L}(y^t)$. Hence, $NSLD(x^t, y^t) = \frac{2 \times \mathcal{L}(y^t)}{2 \times \mathcal{L}(y^t)} = 1$.

D. Proof of Lemma 6

$\mathcal{L}(y^t) \geq \mathcal{L}(x^t)$ is assumed. From Def. 4, $NSLD(x^t, y^t) = \frac{2}{\frac{\mathcal{L}(x^t)+\mathcal{L}(y^t)}{SLD(x^t,y^t)}+1}$. It follows that $\mathcal{L}(y^t) - \mathcal{L}(x^t) \leq SLD(x^t, y^t) \leq \mathcal{L}(y^t)$.

E. Proof of Theorem 2

Trivially, $NSLD(x^t, x^t) = 0$. Moreover, $NSLD(x^t, y^t) = NSLD(y^t, x^t) \forall x^t, y^t$. To prove the Triangle Inequality, let $x^t = \{x^{t1}, x^{t2}, \ldots, x^{tm}\}$, $y^t = \{y^{t1}, y^{t2}, \ldots, y^{tn}\}$, and similarly $z^t = \{z^{t1}, z^{t2}, \ldots, z^{to}\}$. we prove the case $NSLD(x^t, y^t) + NSLD(y^t, z^t) \geq NSLD(x^t, z^t)$.

Because of Lemma 5, if any of x^t, y^t, or z^t are empty, the Triangle Inequality is trivially satisfied. Thus, we consider the case where none of the three tokenized strings are empty.

Again, because of Lemma 5, if $NSLD(x^t, y^t) \geq NSLD(x^t, z^t)$ or $NSLD(y^t, z^t) \geq NSLD(x^t, z^t)$, then the Triangle Inequality is trivially satisfied. Thus, we consider the case where $NSLD(x^t, z^t)$ is greater than both $NSLD(y^t, z^t)$ and $NSLD(x^t, y^t)$.

$$\begin{aligned} &NSLD(x^t, y^t) < NSLD(x^t, z^t), and \\ &NSLD(y^t, z^t) < NSLD(x^t, z^t). \end{aligned} \tag{1}$$

From Rel. 1, we further derive the following relations.

$$\begin{aligned} &\left(\frac{NSLD(x^t, z^t)}{2 - NSLD(x^t, z^t)} - \frac{NSLD(x^t, y^t)}{2 - NSLD(x^t, y^t)}\right) > 0, and \\ &\left(\frac{NSLD(x^t, z^t)}{2 - NSLD(x^t, z^t)} - \frac{NSLD(y^t, z^t)}{2 - NSLD(y^t, z^t)}\right) > 0. \end{aligned} \tag{2}$$

Lemma 4, the Triangle Inequality of SLD, gives the following relations.

$$SLD(x^t, y^t) + SLD(y^t, z^t) \geq SLD(x^t, z^t) \tag{3}$$

From Def. 4, we express SLD in terms of $NSLD$.

$$\begin{aligned} SLD(x^t, y^t) &= \frac{NSLD(x^t, y^t)}{2 - NSLD(x^t, y^t)} \times \left(\mathcal{L}(x^t) + \mathcal{L}(y^t)\right), \\ SLD(y^t, z^t) &= \frac{NSLD(y^t, z^t)}{2 - NSLD(y^t, z^t)} \times \left(\mathcal{L}(y^t) + \mathcal{L}(z^t)\right), and \\ SLD(x^t, z^t) &= \frac{NSLD(x^t, z^t)}{2 - NSLD(x^t, z^t)} \times \left(\mathcal{L}(x^t) + \mathcal{L}(z^t)\right). \end{aligned} \tag{4}$$

Combining Relations 3 and 4, we get the following relations.

$$\begin{aligned} &\frac{NSLD(x^t, y^t)}{2 - NSLD(x^t, y^t)} \times \left(\mathcal{L}(x^t) + \mathcal{L}(y^t)\right) + \\ &\quad \frac{NSLD(y^t, z^t)}{2 - NSLD(y^t, z^t)} \times \left(\mathcal{L}(y^t) + \mathcal{L}(z^t)\right) \geq \\ &\quad\quad \frac{NSLD(x^t, z^t)}{2 - NSLD(x^t, z^t)} \times \left(\mathcal{L}(x^t) + \mathcal{L}(z^t)\right) \end{aligned}$$

This can be rewritten as follows.

$$\begin{aligned} &\left(\frac{NSLD(x^t, y^t)}{2 - NSLD(x^t, y^t)} + \frac{NSLD(y^t, z^t)}{2 - NSLD(y^t, z^t)}\right) \times \mathcal{L}(y^t) \geq \\ &\quad \left(\frac{NSLD(x^t, z^t)}{2 - NSLD(x^t, z^t)} - \frac{NSLD(x^t, y^t)}{2 - NSLD(x^t, y^t)}\right) \times \mathcal{L}(x^t) + \\ &\quad\quad \left(\frac{NSLD(x^t, z^t)}{2 - NSLD(x^t, z^t)} - \frac{NSLD(y^t, z^t)}{2 - NSLD(y^t, z^t)}\right) \times \mathcal{L}(z^t) \end{aligned} \tag{5}$$

From both Lemmas 5 and 6, $1 - \frac{\mathcal{L}(x^t)}{\mathcal{L}(y^t)} \leq NSLD(x^t, y^t) \forall x^t, y^t$. This can be rewritten as the following two relations.

$$\mathcal{L}(x^t) \geq (1 - NSLD(x^t, y^t)) \times \mathcal{L}(y^t) \forall x^t, y^t \tag{6}$$

$$\mathcal{L}(z^t) \geq (1 - NSLD(y^t, z^t)) \times \mathcal{L}(y^t) \forall y^t, z^t \tag{7}$$

Putting together Relations 2, 5, 6, and 7 yields the following.

$$\begin{aligned} &\left(\frac{NSLD(x^t, y^t)}{2 - NSLD(x^t, y^t)} + \frac{NSLD(y^t, z^t)}{2 - NSLD(y^t, z^t)}\right) \times \mathcal{L}(y^t) \geq \\ &\left(\frac{NSLD(x^t, z^t)}{2 - NSLD(x^t, z^t)} - \frac{NSLD(x^t, y^t)}{2 - NSLD(x^t, y^t)}\right) \times \mathcal{L}(y^t) \times \\ &\quad \left(1 - NSLD(x^t, y^t)\right) + \\ &\left(\frac{NSLD(x^t, z^t)}{2 - NSLD(x^t, z^t)} - \frac{NSLD(y^t, z^t)}{2 - NSLD(y^t, z^t)}\right) \times \mathcal{L}(y^t) \times \\ &\quad \left(1 - NSLD(y^t, z^t)\right) \end{aligned} \tag{8}$$

Since $\mathcal{L}(y^t) > 0$, then Rel. 8 can be simplified as follows.

$$\begin{aligned} &\left(\frac{NSLD(x^t, y^t)}{2 - NSLD(x^t, y^t)}\right) \times \left(1 + \left(1 - NSLD(x^t, y^t)\right)\right) + \\ &\left(\frac{NSLD(y^t, z^t)}{2 - NSLD(y^t, z^t)}\right) \times \left(1 + \left(1 - NSLD(y^t, z^t)\right)\right) \geq \\ &\left(\frac{NSLD(x^t, z^t)}{2 - NSLD(x^t, z^t)}\right) \times \left(\left(1 - NSLD(x^t, y^t)\right) + \left(1 - NSLD(y^t, z^t)\right)\right) \end{aligned}$$

This can be further simplified as follows.

$$\begin{aligned} &\frac{\left(NSLD(x^t, y^t) + NSLD(y^t, z^t)\right)}{NSLD(x^t, z^t)} \geq \\ &\quad \frac{2 - \left(NSLD(x^t, y^t) + NSLD(y^t, z^t)\right)}{2 - NSLD(x^t, z^t)} \end{aligned} \tag{9}$$

Rel. 9 implies $NSLD(x^t, y^t) + NSLD(y^t, z^t) \geq NSLD(x^t, z^t)$.

F. Proof of Theorem 3

The proof is by contradiction. W.L.O.G, let $NSLD(x^t, y^t) \leq T$; x^t and y^t has two tokens each, x^{t1}, x^{t2} and y^{t1}, y^{t2}, respectively; and to achieve minimum SLD, x^{ti} is transformed into y^{ti}, $\forall i \in \{1,2\}$. Assuming $NLD(x^{ti}, y^{ti}) > T$, $\forall i \in \{1,2\}$, it follows that $\frac{2 \times LD(x^{ti}, y^{ti})}{|x^{ti}| + |y^{ti}| + LD(x^{ti}, y^{ti})} > T$, $\forall i \in \{1,2\}$. Let $f_1 = 2 \times LD(x^{t1}, y^{t1})$, $f_2 = |x^{t1}| + |y^{t1}| + LD(x^{t1}, y^{t1})$, $f_3 = 2 \times LD(x^{t2}, y^{t2})$, $f_4 = |x^{t2}| + |y^{t2}| + LD(x^{t2}, y^{t2})$. Hence, $\frac{f_1}{f_2} > T$ and $\frac{f_3}{f_4} > T$. Notice that $NSLD(x^t, y^t) = \frac{f_1 + f_3}{f_2 + f_4}$, and that the initial assumption was $NSLD(x^t, y^t) \leq T$, which contradicts $\frac{f_1}{f_2} > T$ and $\frac{f_3}{f_4} > T$. The argument is generalizable for any $\mathcal{T}(x^t)$ and $\mathcal{T}(y^t)$.