PartSS: An Efficient Partition-based Filtering for Edit Distance Constraints

Zhixu Li **Laurianne Sitbon** **Xiaofang Zhou**

School of Information Technology & Electrical Engineering
The University of Queensland, QLD 4072 Australia
Email: {zhixuli, zxf}@itee.uq.edu.au, l.sitbon@uq.edu.au

Abstract

This paper introduces PartSS, a new partition-based filtering for tasks performing string comparisons under edit distance constraints. PartSS offers improvements over the state-of-the-art method NGPP with the implementation of a new partitioning scheme and also improves filtering abilities by exploiting theoretical results on shifting and scaling ranges, thus accelerating the rate of calculating edit distance between strings. PartSS filtering has been implemented within two major tasks of data integration: similarity join and approximate membership extraction under edit distance constraints. The evaluation on an extensive range of real-world datasets demonstrates major gain in efficiency over NGPP and QGrams approaches.

Keywords: edit distance, partition-based, similarity join, approximate membership extraction

1 Introduction

Nowadays, with the fast growing of data sources and the inconsistency between them, our demands for approximate data integration also increase sharply. Two key tasks in data integration are: a) similarity join between data sources to reconcile different representations of the same entities and b) approximate membership extraction, which targets the identification of the substrings from an unstructured text which approximately matches with any member of a given large dictionary of known entities. The entities in the two tasks can be gene or protein names (bioinformatics), geographical locations (geotagging), person or product's names (business information system) *etc.* Due to their importance, similarity join and approximate membership extraction have been studied in many areas such as data cleaning, information integration and bioinformatics.

These two tasks rely on string similarity functions. According to the requirements of different applications, various string similarity functions have been considered. Some applications may prefer to ignore the order of tokens appearing in the text, such as recognizing plagiarisms which share almost the same word set as the original one, or identifying junk mails where some commercial words are frequently used. Therefore, token-based similarity functions such as jaccard similarity, cosine similarity *etc.* are to be used properly (Sarawagi & Kirpal 2004, S. Chaudhuri & Kaushik 2006, Arasu et al. 2006, Bayardo et al. 2007). Other applications are sensitive to token orders, such as retrieving results for typo keywords in information retrieval or recognizing homologous DNA strands in bioinformatics. In these situations, edit distance tends to be a good choice, as it measures the minimum number of edit operations, including *insertion*, *deletion* and *substitution*, to transform one string into another. Several works have used edit distance in similarity join and approximate membership extraction (Zobel & Dart 1995, Bilenko et al. 2003, Gravano et al. 2001, Wang et al. 2009).

Applying edit distance raises great efficiency challenges in Edit Distance similarity Join (EDJ) or Approximate Membership Extraction with Edit Distance threshold (ED-AME) since its computation is expensive($O(n^2)$ time complexity in standard dynamic programming (Wagner & Fischer 1974)). For EDJ between two sets of strings, a naive pairwise similarity join costs $O(N^2 \cdot n^2)$ running time, where N is the number of strings in each set and n is the average length of the strings. For ED-AME from a document D *w.r.t.* a dictionary of string entities R, the naive enumeration method costs $O(|D| \cdot |R| \cdot n^2)$ in running time.

To tackle the low efficiency problem brought by computing edit distance, several approaches have been proposed that raise the efficiency by pruning unmatched pairs of strings with filtering techniques. A widely adopted filter is the conversion of the edit distance constraint between two strings into the overlap between their q-grams sets (Gravano et al. 2001). The main inconvenient of this approach is the sensitivity to the choice of the size of q-grams. Another line of works are developing filtration based on finding common neighbors between strings. Bocek *et. al.* proposed to generate τ-*deletion neighborhood* sets by deleting no more than τ characters at all possible positions of a string, where τ is the given edit distance threshold. If two strings are within τ edit distance to each other, they must share at least one τ-*deletion neighborhood.* This filtering may only work well for short strings with a small edit distance threshold τ, otherwise the size of *deletion neighborhood* $O(|s|^\tau)$ can be large, where $|s|$ denotes the length of the string s. To further reduce the neighborhood size, an improved method called NGPP (stands for Neighborhood Generation with partitioning and prefix-based pruning) was proposed by Wang *et. al.* (Wang et al. 2009). NGPP first partitions a string s into several partitions, then slightly adjusts the position and length of these partitions to generate variants for each partition. It is guaranteed that for any string s' within τ edit distance to s there must be at least one partition of s' within 1 edit distance to a variant of the partition which lies at the same position in s. Only 1-*deletion neighborhoods* are generated for each partition variant of s and the overall neighborhood size of s is $O(|s|\tau + \tau^2)$.

NGPP greatly decreases the time and space complexity of EDJ or ED-AME, but it still need to generate 1-*deletion neighborhoods* and the neighborhood size also becomes large when $|s|$ is a long string. In this paper, we propose a novel partition-based filtering technique to be implemented with EDJ and ED-AME. The new partition scheme *PlusOne* and partition variation generation rule are proposed, leading to the generation of $O(\tau^2)$ partition variants for string s, such that for any string s' that within τ edit distance to s, there must be at least one partition of s' equals to a variant of the partition which lies at the same position of s. In order to prevent the generation of unnecessary partition variants, we also develop some non-trivial techniques to give a stricter variation generation rule for our method. By applying the prefix-based filtering (Wang et al. 2009), the size of partition signatures can be further reduced to $O(l_p \cdot \tau)$, where l_p is the length of prefix set in the prefix-based pruning.

Our main contributions in this paper are summarized below:

- We propose a novel partition-based filter for EDJ and ED-AME using a new partition scheme *PlusOne* reducing the space complexity to $O(\tau^2)$ in addition to reducing the probability of false-positives.
- We develop a strict partition variant generation rule, which gives tighter upper-bounds to the range of *shifting* and *scaling* operations applied to partitions of the string.
- We apply this filter into both EDJ and ED-AME demonstrate the effectiveness of the filtration based on our partition-based scheme on several real-world datasets.

The outline of this paper is as follows: Section 2 covers related work. We introduce preliminaries in Section 3. Section 4 presents the novel partition-based signature scheme for edit distance. We then introduce how we adapt our scheme into EDJ and ED-AME in Section 5 and 6. The experimental study is given in Section 7.

2 Related Work

Edit distance is a widely-used distance function for strings, which can be computed in $O(n^2)$ time and $O(n)$ space using the standard dynamic programming (Wagner & Fischer 1974). In order to enhance the computing efficiency, some techniques have been proposed in the past several decades, such as the Four-Russians (Masek & Paterson 1980) which improves the time complexity into $O(n^2/log(n))$, or a bit-parallel algorithm (Ukkonen 1983) which reaches an average time complexity of $O(n + d^2)$ (d is the edit distance between two strings).

Recently, edit distance was introduced into the Similarity Join (SJ) and Approximate Membership Extraction (AME) (Zobel & Dart 1995, Bilenko et al. 2003, Gravano et al. 2001, Wang et al. 2009), making efficiency problems arise. The state-of-art approaches to process edit similarity join or edit approximate membership extraction are mainly based on a popular filtration-verification framework. In the filtration step, a great amount of string pairs (or substring and entity pairs) are pruned and the similarity of the remaining pairs is calculated in the verification step,

A popular filtering scheme is to convert the edit distance constraint into a weaker *count filtering* on the number of matching q-grams. Q-grams are generated by sliding a window of width q over a string. If two strings s and t are within edit distance τ, they must share at least $LB_{s,t} = max(|s|, |t|) - q + 1 - q \cdot \tau$ q-grams (Gravano et al. 2001). Other methods were also suggested to complement the count filtering. *length filtering* guarantees that s and t's length disparity should be within τ. *Position filtering* requires that there are at least $LB_{s,t}$ *matching positional* q-grams (Gravano et al. 2001). Since generating all pairs of strings satisfying the count filtering rule is a bottleneck, the *prefix filtering* strategy was used later to quickly discard some unmatched pairs without accessing all of their q-grams (S. Chaudhuri & Kaushik 2006). As state-of-the-art techniques, Xiao *et. al.* proposed *location-based mismatch filtering* and *content-based mismatch filtering*, which can be incorporated into previous strategies to make the edit similarity join be more efficient (Xiao et al. 2008). However, it is a dilemma to choose proper length of q-grams when using q-gram-based approaches to find approximate matches. On the one hand, q should be smaller than $\frac{L_{min}+1}{\tau+1}$ (L_{min} is the shortest length of entity string) to guarantee at least one common q-gram between matching strings. On the other hand, short q-grams lead to long posting lists impacting the overall cost of similarity join (Xiao et al. 2008, Zobel & Dart 1995). Besides, q-gram-based approaches are less selective for short entities, and it is hard to share computation (Wang et al. 2009).

Both SJ and AME have been studied extensively in the literature. SJ is related to record linkage (Winkler 1999), name matching (Bilenko et al. 2003), data deduplication (Sarawagi & Bhamidipaty 2002) *etc.* AME is also known as Membership Checking or Approximate Entity Extraction. Besides edit distance, various alternative similarity functions have been used in SJ and AME. These works are also mainly based on the filtration-verification framework. In the filtration step, various kinds of signature schemes (S. Chaudhuri & Kaushik 2006, Chandel et al. 2006, Arasu et al. 2006, Gionis et al. 1999), or inverted list index (Chandel et al. 2006, Singhal 2001), or combinations (Chakrabarti et al. 2008, Lu et al. 2009, Wang et al. 2009) have been proposed.

Also related field of SJ and AME, "approximate string matching" focuses on finding a pattern string approximately in a text. This problem has been studied extensively in algorithmic, information retrieval and natural language process communities. An excellent survey is given in (Navarro 2001).

3 Preliminaries

Our proposed method builds on previous filtering approaches that we briefly review in this section.

3.1 Neighborhood Generation-based filtering

A series of approaches based on neighborhood generation has been developed to deal with edit distance constraints. All the strings that are at most at τ edit distance away from a string s constitute the τ neighborhood of s, formally represented as $U_\tau(s) = \{s' | ed(s, s') \leq \tau\}$, where $ed(s, s')$ is the edit distance between s and s'. Therefore, if string t can approximate match with s, it must be contained in the τ neighborhood of s. However, since the size of the neighborhood $U_\tau(s)$ is $O(|s|^\tau \cdot |\sum|^\tau)$ in practice, it is impossible to generate and use all the neighborhood directly.

The *deletion neighborhood* was proposed as a complement (Bocek et al. 2007). The τ-deletion neighborhood of a string s consists of all deletion variants of s, each of which is generated by deleting no more than τ characters at all possible positions of s. If another string t is within the edit distance τ of s, then they share at least one common variant in their τ deletion neighborhood. Based on the property of *deletion neighborhood*, the FastSS algorithm proposed in (Bocek et al. 2007) successfully reduces the size of the neighborhood to $O(|s|^\tau)$. However this algorithm is suitable for short strings with a small edit distance threshold τ.

3.2 NGPP

The improved neighborhood generation-based method NGPP has been proposed (Wang et al. 2009) to further reduce the size of the deletion neighborhood to $O(l_p \cdot \tau^2)$, where l_p is the length of the prefix set in the prefix-based pruning. Now we briefly introduce NGPP.

The partition scheme defined in NGPP partitions a string into $k_\tau = \lfloor \frac{\tau+1}{2} \rfloor$ partitions, with the first $k_\tau - 1$ partitions of length $\lfloor \frac{m}{k_\tau} \rfloor$, and the k_τ (the last) partition taking the rest of the string, where m is the length of the string. For a string s and any of its τ-neighborhood string s' (a string within τ edit distance of s), at least one partition of s, subject to appropriate amount of shifting and scaling operation, is within edit distance 1 to the partition at the same position of s'. The shifting and scaling operations are defined to generate variations for partitions of s.

- **Shifting** by a moves the partition $s[i...j]$ by a positions to $s[(i+a)...(j+a)]$
- **Scaling** by b changes the length of $s[i...j]$ by b to $s[i...(j+b)]$.

Table 1 presents the constraints on the shifting and scaling ranges for each partition of string s, according to the possible length and position dispatch between s and its τ-neighborhood string.

Table 1: Shifting and Scaling Range for Each Partition in NGPP

Partition	Shifting Range	Scaling Range
the first one	0	$[-2, 2]$
the last one	$[-\tau, \tau]$	equals to shifting amount
others	$[-\tau, \tau]$	$[-2, 2]$

Once partition variants are generated for a string s, 1-deletion neighborhood variants are generated for each partition variant. Thus $O(\tau \cdot |s| + \tau^2)$ deletion neighborhood are generated for s. In order to measure if another string s' is a τ-neighborhood string of s, all partition variations $(\lfloor \frac{|t|}{k_\tau} - 2 \rfloor)$ are generated for s'.

Example 1 *Given a string s = "Leonardo Dicaprio" and a misspell string t = "Leeonardo Diecabrio". When $\tau = 3$, we have $k_\tau = 2$. According to the partition scheme, s and s' can be partitioned as follows, where we use # to represent the blank between words:*

s={[Leonardo], [#Dicaprio]}
s'={[Leeonardo#], [Diecabrio]}

The 1-deletion variants for all partitions of s are:

⟨[Leonar], 1⟩	⟨[Leonard], 1⟩
⟨[Leonardo], 1⟩	⟨[Leonardo#], 1⟩
⟨[Leonardo#D], 1⟩	⟨[rdo#Dicaprio], 2⟩
⟨[do#Dicaprio], 2⟩	⟨[o#Dicaprio], 2⟩
⟨[#Dicaprio], 2⟩	⟨[Dicaprio], 2⟩
⟨[icaprio], 2⟩	⟨[caprio], 2⟩

Since the first partition of s', [Leeonardo#], shares a 1-deletion variant [Leonardo#] with s's partition variation [Leonardo#], s' is within τ edit distance from s.

When s and s' are long, not only space complexity increases to store the variations of s, but also time complexity in matching partition variants of s' against those of s. In order to reduce these complexities, a prefix-based pruning was introduced. When a partition is longer than a prefix length l_p, only 1-deletion neighborhood variants are generated for the l_p-prefix of the partition.

Example 2 *Continuing with Example 1, With $l_p = 3$, the following 1-deletion variants are generated from the prefixes :*

⟨[Leo], 1⟩	⟨[rdo], 2⟩	⟨[do#], 2⟩
⟨[o#D], 2⟩	⟨[#Di], 2⟩	⟨[Dic], 2⟩
⟨[ica], 2⟩	⟨[cap], 2⟩	

For each string s', a neighborhood of $O(l_p \cdot \tau^2)$ variants are generated and searched against all variants of s. The approach we propose in this paper is another pure partition-based approach without generating deletion variants, which only generates $O(|s| + \tau^2)$ partition variants. Combining with prefix-based method, the size of variants can be further reduced to $O(l_p \cdot \tau)$.

3.3 PartEnum

PartEnum is another partition-based approach proposed for edit distance constraints (Arasu et al. 2006). Based on the pigeon hole principle, it first divides a string into several partitions, then generates signatures for each partition using an enumeration scheme. This scheme guarantees that if two strings are matched within a given edit distance threshold, they must share at least one signature. It shows good performances for small edit distance threshold. However, since the performance of PartEnum is greatly dependent on the partitioning parameters, it is hard to set the parameters that work well for both short and long entities (Wang et al. 2009).

4 PartSS: A New Partition-Based Filtering

In this section, we introduce a new partition-based filter – PartSS, which stands for Partition-based filter with Shifting and Scaling operations. We present our partition scheme *PlusOne* firstly, then develop some non-trivial techniques to tighten the partition variants set.

4.1 Partitioning

The principle of PartSS is directly inspired by that of NGPP. By setting the partition number to k disjoint partitions, according to the pigeon hole principle, there exists at least one partition which only need at most $\lfloor \frac{\tau}{k} \rfloor$ edit operations for changing s into any other string s' which edit distance to s is within τ. In NGPP $k_\tau = \lceil \frac{\tau+1}{2} \rceil$. In order to avoid the generation of a deletion neighborhood, we choose to set $k_\tau = \tau + 1$, so that there exists at least one partition in s, after subjecting to some shifting and scaling, that directly equals to the partition in the same position of s'.

PlusOne Partition Scheme: Given an edit distance threshold τ, *PlusOne* partition scheme partitions a string s into $k_\tau = \tau + 1$ partitions, where the first $n_1 = k_\tau - |s|\%k_\tau$ partitions have length $l_1 = \lfloor \frac{|s|}{k_\tau} \rfloor$, and the other $n_2 = |s|\%k_\tau$ partitions have length $l_2 = \lceil \frac{|s|}{k_\tau} \rceil$, such that $n_1 \cdot l_1 + n_2 \cdot l_2 = |s|$. (As shown in Fig. 1)

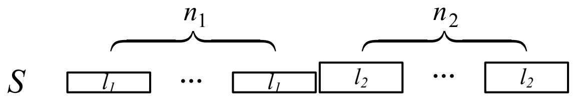


Figure 1: The Partitions of String s under PlusOne Scheme

Due to the length and position deviation between corresponding partitions in s and s', we have to adjust the length and position of partitions in s. Here we adopt the two operations (shifting and scaling) defined in NGPP, as introduced in Section 3. Under *PlusOne*, the *baseline* rule on how to apply shifting and scaling operations to every partition of a string is given below and summarised in Table 2. This baseline rule is similar to the one used in NGPP (Wang et al. 2009).

Rule 1 ***(Baseline Variants Generation Rule)*** *Given the edit distance threshold τ, a string s is partitioned into $\tau+1$ partitions under the PlusOne partitioning scheme. Then for the i-th partition of s:*

- *if $i = 1$, we scale the partition within the range of $[-1, 1]$ to generate its variants.*
- *if $i = \tau + 1$, we scale and shift the partition to the same amount within the range of $[-\tau, \tau]$ to generate its variants.*
- *if $1 < i < \tau$, we scale the partition within the range of $[-\tau, \tau]$, while also shifting it within the range of $[-2, 2]$, to generate its variants.*

Table 2: Baseline Shifting and Scaling Rule for Each Partition

Partition	Shifting Range	Scaling Range
the first one	0	$[-1, 1]$
the last one	$[-\tau, \tau]$	equals to shifting amount
others	$[-\tau, \tau]$	$[-1, 1]$

Based on our proposed *PlusOne* partitioning scheme and partition variants generation rules, the partitions have the following properties:

Property 1 ***(Partition Variants Property)*** *Given a string s and an edit distance threshold τ, if we partition s with PlusOne partitioning scheme and generate partition variants according to the baseline rule above, for any string s' which satisfy $ed(s, s') \leq \tau$, there must be at least one partition in a position of s' (also partitioned with PlusOne) matching with a variant of the partition in the same position of s.*

As mentioned in Section 3, NGPP generates $O(m \cdot \tau + \tau^2)$ variants for a string, whereas the baseline rule based on *PlusOne* reduces the size of variants to $O(\tau^2)$.

4.2 Tightening the Shifting and Scaling Ranges

In the baseline rule, the boundaries of the shifting and scaling ranges are set very loose to guarantee full coverage. On the other hand, loose boundaries also lead to more false-positives, since more unpromising matches will be considered as candidate matches. In order to give a *minimum coverage* of all τ-neighborhood, we expect to find a stricter rule to tighten the boundaries of shifting and scaling ranges, such that unnecessary variants of partitions are not generated. Meanwhile, this new rule should also satisfy Property 1.

For convenience of presentation, we differentiate the original shifting and scaling operations into four different operations: negative shifting, positive shifting, negative scaling and positive scaling. The shifting or scaling amount of any of the four operations is a non negative numeric referred to as the *variation amount* of that operation.

In order to achieve *minimum coverage*, we give different upper-bounds to the *variation amount* of each operation applied to partitions at different positions. A straightforward approach is to enumerate all τ-neighborhoods of the string s: for any neighborhood string s', there must exists one partition of s that requires the least *variation amount* of an operation to become the corresponding partition of s'. We call this partition as the *key partition* in transforming from s to s' (denoted $p_{key}^{s \to s'}$). When there are more than one key partitions in a transformation, only the one at the first position is considered. To reach a minimum coverage, we deem that only the operations applied on key partition in each transformation are necessary. Accordingly, the following property on the tightest upper-bound of operations to each partition holds:

Property 2 ***(Tightest upper-bound Property)*** *Given a string s, the tightest upper-bound of an operation on a partition of s is the maximum variation amount of the operation to that partition when it acts as a key partition.*

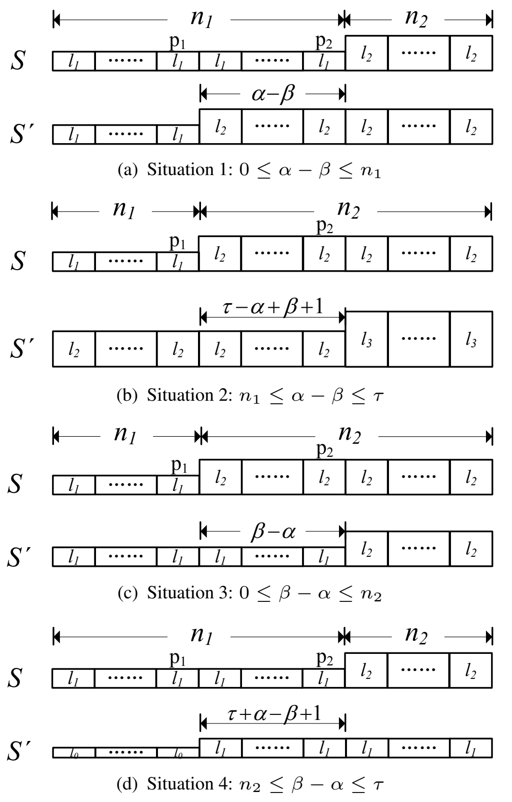
Although it is impractical to enumerate all τ-neighborhoods of s, they can be classified into four types according to their length and position deviation to s, and we can discuss the upper-bounds of the *variation amount* of the four operations in the four different transformation situations below:

Four Transformation Situations: Given the edit distance threshold τ, assume we need α *insertion*, β *deletion* and γ *substitution* to transform s into a $\tau-$neighborhood string s' such that $\alpha + \beta + \gamma \leq \tau$. Since the *insertion* and *deletion* operations introduce position and length changes, all strings within τ edit distance to s can be classified into one of the four situations depending on the value of α and β. Based on the *PlusOne* partition scheme, at most two kinds of partitions with different length are generated for each string. Here we assume the first n_1 partitions have length l_1, and the remaining n_2 partitions have length l_2 in s. According to the *PlusOne* scheme, $l_1 + 1 = l_2$, $n_1 + n_2 = \tau + 1$. We also define $l_0 = l_1 - 1$, $l_3 = l_2 + 1$, which are possible lengthes of partitions in τ-neighborhood strings as shown in Fig. 2.

- Situation 1: If $0 \leq \alpha - \beta \leq n_1$, then the first $n_1 - (\alpha - \beta)$ partitions have length l_1, and the other ones have length l_2;
- Situation 2: If $n_1 \leq \alpha - \beta \leq \tau$, then the first $n_1 + (\tau - \alpha + \beta + 1)$ partitions have length l_2, and the other ones have length l_3;
- Situation 3: If $0 \leq \beta - \alpha \leq n_2$, then the first $n_1 + (\beta - \alpha)$ partitions have length l_1, and the other ones have length l_2;
- Situation 4: If $n_2 \leq \beta - \alpha \leq \tau$, then the first $n_1 + (\tau + \alpha - \beta + 1)$ partitions have length l_0, and the other ones have length l_1.

In the remainder of this section, we use $\sigma_+(i)$ and $\sigma_-(i)$ to denote the *variation amount* of the positive and negative shifting operations and $\omega_+(i)$ and $\omega_-(i)$ to denote the *variation amount* of positive and negative scaling operations that have to be applied to the i-th partition. We also use $\overline{\sigma_+}(i)$, $\overline{\sigma_-}(i)$, $\overline{\omega_+}(i)$, $\overline{\omega_-}(i)$ to denote the upper-bounds of their *variation amounts*. It is worth noting that: $\overline{\omega_+}(i)$ only needs to be discussed in situation 1 and 2, while $\overline{\omega_-}(i)$ only needs to be discussed in situation 3 and 4. We only provide the detailed deduction process in situation 1 as the deduction process for the other three situations follows the same principle.

Situation 1: $0 \leq \alpha - \beta \leq n_1$, since $\alpha + \beta <= \tau$, we have: $0 \leq \alpha \leq \frac{\tau + n_1}{2}$. According to the difference between partition lengthes at the same position of s and s', the partition set can be divided into three disjoint consecutive parts. We use p_1 and p_2 to denote the last partition



n_1
n_2
p_1
p_2
S
S'
l_1
l_2
l_3
$\alpha-\beta$
$\tau-\alpha+\beta+1$
$\beta-\alpha$
$\tau+\alpha-\beta+1$
(a) Situation 1: $0 \le \alpha - \beta \le n_1$
(b) Situation 2: $n_1 \le \alpha - \beta \le \tau$
(c) Situation 3: $0 \le \beta - \alpha \le n_2$
(d) Situation 4: $n_2 \le \beta - \alpha \le \tau$

Figure 2: The Transformation from s to s' in Four Situations

position within the first and second parts respectively such that: $0 \le p_1 = n_1 - (\alpha - \beta) \le n_1$ and $p_2 = n_1$.

Positive scaling $\omega_+(i)$: We can easily observe the length variation for partitions of three different parts in Figure 2. Equation 1 gives the value of $\overline{\omega_+}(i)$ for the i-th partition.

$$\overline{\omega_+}(i) = \begin{cases} 0 & (1 \le i \le p_1) \\ 1 & (p_1 + 1 \le i \le n_1) \\ 0 & (n_1 + 1 \le i \le \tau + 1) \end{cases} \tag{1}$$

Positive shifting $\sigma_+(i)$: Equation 2 gives the maximum value of $\sigma_+(i)$ for the i-th partition, where α_{pre} denotes the *insertions* applied to the partitions before the i-th one.

$$\overline{\sigma_+}(i) = \begin{cases} 0 & (i = 1) \\ \alpha_{pre} & (2 \le i \le p_1) \\ \alpha_{pre} - i + p_1 + 1 & (p_1 + 1 \le i \le n_1) \\ \alpha_{pre} + p_1 - n_1 & (n_1 + 1 \le i \le \tau + 1) \end{cases} \tag{2}$$

Since the function given in Equation 2 is declining, the condition on the i-th partition acting as the key partition is that at least one edit operation should be applied to each partition after the i-th one (except for those whose variation amount of positive scaling equals to that of the i-th one). Given $\alpha_{pre} \le \alpha \le \frac{\tau+n_1}{2}$, the maximum value of α_{pre} is:

$$\alpha_{pre} = \begin{cases} min(p_1, \frac{\tau+n_1}{2}) = p_1 & (2 \le i \le p_1) \\ min(i-1, \frac{\tau+n_1}{2}) = i - 1 & (p_1 + 1 \le i \le n_1) \\ min(\tau, \frac{\tau+n_1}{2}) = \frac{\tau+n_1}{2} & (n_1 + 1 \le i \le \tau + 1) \end{cases} \tag{3}$$

Considering the maximum value of α_{pre}, the maximum value of $\sigma_+(i)$ is:

$$\overline{\sigma_+}(i) = \begin{cases} 0 & (i = 1) \\ p_1 & (2 \le i \le p_1) \\ p_1 & (p_1 + 1 \le i \le n_1) \\ \frac{\tau+n_1}{2} & (n_1 + 1 \le i \le \tau + 1) \end{cases} \tag{4}$$

Given the range of p_1, the range of $\sigma_+(i)$ or $\sigma_+(i)$ and its corresponding value of $\omega_+(i)$ is in Table 3 and 4.

Table 3: $\sigma_+(i)$ and corresponding $\omega_+(i)$ I

***i*-th Partition**	$\sigma_+(i)$	$\overline{\omega_+}(i)$
$i = 1$	0	0
$2 \le i \le n_1$	$[0, n_1]$	0
$1 \le i \le n_1$	$[0, i-1]$	1
$n_1 + 1 \le i \le \tau + 1$	$[0, \frac{\tau+n_1}{2}]$	0

Table 4: $\sigma_+(i)$ and corresponding $\omega_+(i)$ in situation 1

***i*-th Partition**	$\sigma_+(i)$	$\overline{\omega_+}(i)$
$i = 1$	0	1
$2 \le i \le n_1$	$[0, i-1]$	1
$2 \le i \le n_1$	$[i, n_1]$	0
$n_1 + 1 \le i \le \tau + 1$	$[0, \frac{\tau+n_1}{2}]$	0

Negative shifting $\sigma_-(i)$: Equation 5 gives the maximum value of $\sigma_-(i)$ for the i-th partition, where β_{pre} denotes the *deletions* applied to the partitions before the i-th one.

$$\overline{\sigma_-}(i) = \begin{cases} 0 & (i = 1) \\ \beta_{pre} & (2 \le i \le p_1) \\ \beta_{pre} + i - p_1 - 1 & (p_1 + 1 \le i \le n_1) \\ \beta_{pre} - p_1 + n_1 & (n_1 + 1 \le i \le \tau + 1) \end{cases} \tag{5}$$

Since the function given in Equation 5 is increasing, the condition on the i-th partition acting as the key partition is that at least one edit operation is applied to each partition before the i-th one, which do not give any constraint on the value of β_{pre}. Given $\beta_{pre} \le p_1 - p_2 + \alpha \le \frac{\tau-n_1}{2} + p_1$, the maximum value of β_{pre} is $\frac{\tau-n_1}{2} + p_1$.

Considering the maximum value of β_{pre}, the maximum value of $\sigma_-(i)$ is:

$$\overline{\sigma_-}(i) = \begin{cases} 0 & (i = 1) \\ \frac{\tau-n_1}{2} + p_1 & (2 \le i \le p_1) \\ \frac{\tau-n_1}{2} + i - 1 & (p_1 + 1 \le i \le n_1) \\ \frac{\tau+n_1}{2} & (n_1 + 1 \le i \le \tau + 1) \end{cases} \tag{6}$$

Given the range of p_1, the range of $\sigma_-(i)$ or $\sigma_-(i)$ and its corresponding value of $\omega_+(i)$ is in Table 5 and 6.

Table 5: $\sigma_-(i)$ and corresponding $\omega_+(i)$ I

***i*-th Partition**	$\sigma_-(i)$	$\overline{\omega_+}(i)$
$i = 1$	0	0
$2 \le i \le n_1$	$[0, \frac{\tau+n_1}{2}]$	0
$1 \le i \le n_1$	$[0, \frac{\tau-n_1}{2} + i - 1]$	1
$n_1 + 1 \le i \le \tau + 1$	$[0, \frac{\tau+n_1}{2}]$	0

Summary: Following the same analysis to generate correspondences values for situations 2, 3 and 4, the overall tight partition variants generation rules are deducted as presented in Table 7-10. The four tables give different upper-bounds to the *variation amount* of each operation applied to partitions at at different positions.

Table 6: $\sigma_-(i)$ and corresponding $\omega_+(i)$ in situation 1

i**-th Partition**	$\sigma_-(i)$	$\overline{\omega_+}(i)$
$i = 1$	0	1
$2 \leq i \leq n_1$	$[0, \frac{\tau - n_1}{2} + i - 1]$	1
$2 \leq i \leq n_1$	$[\frac{\tau - n_1}{2} + i, \frac{\tau + n_1}{2}]$	0
$n_1 + 1 \leq i \leq \tau + 1$	$[0, \frac{\tau + n_1}{2}]$	0

Table 7: $\sigma_+(i)$ and corresponding $\omega_+(i)$

i**-th Partition**	$\sigma_+(i)$	$\overline{\omega_+}(i)$
$i = 1$	0	1
$2 \leq i \leq n_1$	$[0, i - 1]$ $[i, n_1]$	1 0
$i = n_1 + 1$	$[0, \frac{\tau + n_1}{2}]$	0
$n_1 + 2 \leq i \leq \tau + 1$	$[0, i - n_1 - 1]$ $[i - n_1, \frac{\tau + n_1}{2}]$	1 0

Table 8: $\sigma_+(i)$ and corresponding $\omega_-(i)$

i**-th Partition**	$\sigma_+(i)$	$\overline{\omega_-}(i)$
$i = 1$	0	0
$2 \leq i \leq n_1$	$[0, \frac{\tau}{2}]$	1
$n_1 + 1 \leq i \leq \tau + 1$	$[0, \frac{\tau}{2} + i - n_1 - 1]$ $[\frac{\tau}{2} + i - n_1, \frac{n_1 - 3}{2} + i]$	1 0

Table 9: $\sigma_-(i)$ and corresponding $\omega_+(i)$

i**-th Partition**	$\sigma_-(i)$	$\overline{\omega_+}(i)$
$i = 1$	0	1
$2 \leq i \leq n_1$	$[0, \tau - n_1 + i - 1]$ $[\tau - n_1 + i, \frac{\tau + n_1}{2}]$	1 0
$i = n_1 + 1$	$[0, \tau]$	0
$n_1 + 2 \leq i \leq \tau + 1$	$[0, i - 2]$ $[i - 1, \tau]$	1 0

Table 10: $\sigma_-(i)$ and corresponding $\omega_-(i)$

i**-th Partition**	$\sigma_-(i)$	$\overline{\omega_-}(i)$
$i = 1$	0	1
$2 \leq i \leq n_1$	0 $[1, min(n_1, \tau - \frac{n_1 - 1}{2})]$	1 0
$n_1 + 1 \leq i \leq \tau + 1$	$[0, n_1]$ $[n_1 + 1, \tau + \frac{n_1 + 3}{2} - i]$	1 0

Example 3 *Based on the same string and setting as Example 1, when $\tau = 3$, $k_\tau = 4$, according to the partition scheme, s and t can be partitioned into:*

s={[Leon], [ardo], [#Dic], [aprio]}
t={[Le$\underline{e}$o], [nardo], [#Di$\underline{e}$c], [a$\underline{b}$rio]}

In the next step, shifting and scaling operations should be applied to the partitions of s to generate variants. Through calculation, we have the tightened shifting and scaling ranges for partitions of s as shown in Table 11.

The second partition of t is nardo, and can be found in the variants of the second partition of s. Therefore, t passes the filtration and becomes a candidate string within the 3 edit distance to s.

4.3 Combining with Prefix-based Filtering

The size of the variants set can be further reduced by adopting the prefix-based filtering used in (Wang et al. 2009). A fix length l_p prefix of each variant can reduce the number of the variants without leading to any false-negatives. By applying the prefix-based filtering (Wang et al. 2009), the size of partition signatures can be further reduced to $O(l_p \cdot \tau)$.

Table 11: All Variants of Partitions in "*Leonardo Dicaprio*"

Partition	**Variants**
$[Leon]$	$\langle Leo, 0\rangle$, $\langle Leon, 0\rangle$, $\langle Leona, 0\rangle$
$[ardo]$	$\langle ona, 1\rangle$, $\langle onar, 1\rangle$, $\langle onard, 1\rangle$, $\langle nar, 1\rangle$, $\langle nard, 1\rangle$, $\langle nardo, 1\rangle$, $\langle ard, 1\rangle$, $\langle ardo, 1\rangle$, $\langle ardo\#, 1\rangle$, $\langle rdo, 1\rangle$, $\langle rdo\#, 1\rangle$, $\langle rdo\#D, 1\rangle$, $\langle do\#, 1\rangle$, $\langle do\#D, 1\rangle$, $\langle do\#Di, 1\rangle$, $\langle o\#D, 1\rangle$, $\langle o\#Di, 1\rangle$, $\langle o\#Dic, 1\rangle$
$[\#Dic]$	$\langle do\#D, 2\rangle$, $\langle do\#Di, 2\rangle$, $\langle o\#Di, 2\rangle$, $\langle o\#Dic, 2\rangle$, $\langle \#Dic, 2\rangle$, $\langle \#Dica, 2\rangle$, $\langle Dica, 2\rangle$, $\langle Dicap, 2\rangle$, $\langle icap, 2\rangle$, $\langle icapr, 2\rangle$, $\langle capr, 2\rangle$, $\langle capri, 2\rangle$
$[aprio]$	$\langle Dicaprio, 3\rangle$, $\langle icaprio, 3\rangle$, $\langle caprio, 3\rangle$, $\langle aprio, 3\rangle$, $\langle prio, 3\rangle$, $\langle rio, 3\rangle$, $\langle io, 3\rangle$

Example 4 *Continuing with example 3, by setting the prefix length to $l_p = 3$, we have the prefix of variants of s as shown in Table 12.*

Table 12: Prefix Variants of "*Leonardo Dicaprio*"

Partition	**Variants**
$[Leon]$	$\langle Leo, 0\rangle$
$[ardo]$	$\langle ona, 1\rangle$, $\langle nar, 1\rangle$, $\langle ard\#, 1\rangle$, $\langle rdo, 1\rangle$, $\langle do\#, 1\rangle$, $\langle o\#D, 1\rangle$
$[\#Dic]$	$\langle do\#, 2\rangle$, $\langle o\#D, 2\rangle$, $\langle \#Di, 2\rangle$, $\langle Dic, 2\rangle$, $\langle ica, 2\rangle$, $\langle cap, 2\rangle$
$[aprio]$	$\langle Dic, 3\rangle$, $\langle ica, 3\rangle$, $\langle cap, 3\rangle$, $\langle apr, 3\rangle$, $\langle pri, 3\rangle$, $\langle rio, 3\rangle$, $\langle io, 3\rangle$

5 Implementing PartSS Filtering in EDJ

Given two sets of strings R and S, the task of edit similarity join (or similarity join with an edit distance threshold τ) is to find all pairs of strings (r, s) ($r \in R$ and $s \in S$) such that their edit distance is no larger than the given threshold τ, ie., $\{(r, s)|ed(r, s) \leq \tau, r \in R, s \in S\}$.

To avoid pairwise comparison between two string sets R and S, an inverted list is usually leveraged, with one set of strings indexed with their signatures in the inverted list. In this inverted list, a signature sig is mapped to a posting list I_{sig}, in which each entity records the id and other information about the string which has sig in its signature set. Strings in the other set are taken as queries to search against the inverted list. Candidate matches of a query string are retrieved by generating signatures for the query string and searching them against the inverted list. Final results can be found by verifying the true edit distance between the query string and retrieved ones. Filtering techniques can be applied during this query and candidate generation process.

We observe that most signature schemes (including PartEnum, Q-gram based) are symmetrical in applying the same signature generation method to both the indexed strings and the query strings. PartSS asymmetrically generates partition variants for indexed strings with both *PlusOne* partition scheme and partition variant generation scheme, then only generates partitions for query strings with the *PlusOne* partition scheme. In the inverted list, each entry in the posting list I_{sig} is in the form of (rid, pid), where rid refers to the id of the string which has sig as one of its partition variants, and pid refers to the position of the partition which has sig as one of its variants in rid string. Then for each query string s, $\tau+1$ positional partitions are generated and searched against the inverted list, with each partition in the form of $(partition, pid')$. The retrieved indexed strings which satisfy the following two conditions become candidate matches of s:

- *Positional Partition Filtering:* at least one partition (variant) should be in the same position as the retrieved index string and the string s, ie., $pid = pid'$.
- *Length Filtering:* the deviation between the length of the retrieved partition and that of s should be within τ.

6 Implementing PartSS Filtering in ED-AME

Given a dictionary of string entities R, for an input document D, the task of approximate membership extraction with edit distance threshold τ is to find all substrings from D within τ edit distance from one entity in R, ie., $\{D[i...j] \mid \exists r \in R, ed(D[i...j], r) \leq \tau\}$, where $D[i...j]$ represents a substring located at positions from i to j in D.

A straightforward way to apply PartSS in ED-AME is to index the dictionary R in an inverted list in the same way as we do in EDJ, then enumerate all the possible substrings within the length $[L_{min} - \tau, L_{max} + \tau]$ (L_{min} and L_{max} represent the shortest and longest length of entities in R) from the given document D as query segments. Candidate entities can be retrieved by probing each query segment against the inverted list. The final results of ED-AME can be obtained by verifying all query segments and entity pairs. Although promising a full coverage, this algorithm inefficiently exhausts too many query segments, some of which have no match at all.

A more efficient query document processing algorithm was proposed and combined with NGPP by Wang *et. al.* (Wang et al. 2009), which can jump across many unnecessary substring and entity pairs by leveraging fixed prefix of partitions. Here we briefly introduce how to combine this document processing algorithm with PartSS. In this algorithm: all dictionary entities are classified into long entities and short entities and indexed into two inverted index I^{long} and I^{short} respectively. For each long entity no shorter than $k_\tau \cdot l_p$, we generate partition variants by the *PlusOne* and Variant Generation rule, then all l_p-prefixes of partitions are generated and then indexed into I^{long}. For each short entity shorter than $k_\tau \cdot l_p + \tau$, a prefix of $min(|e|, l_p)$ is taken to generate its $\tau-$variant deletion neighborhood (see section 3.1), which are then indexed into I^{short} (note that entities between $k_\tau \cdot l_p$ and $k_\tau \cdot l_p + \tau$ are mapped into both indexes to ensure no match will be missing in the document processing (Wang et al. 2009)).

Once all dictionary entities are indexed, for a query document D, we enumerate all l_p-length segments, denoted as $\{seg = D[p...p + l_p - 1] \mid 0 \leq p \leq |D| - L_{min} + \tau + 1\}$, and search them against the two inverted indexes in the following manner.

- *Searching against Long Index:* we use seg to search against the long index I^{long} to find all entities which have seg as a prefix of a partition variant at any position of the entity. Then for each entity e in the retrieved candidate set E, assume seg is the prefix of variant of its pid-th partition, such that e might match with any substring which has seg as prefix of its pid-th partition in D. To locate these substrings in D, for any possible length m within $[|e| - \tau, |e| + \tau]$ (see the Length Filtering in section 5.2), the partition size can be decided as $\lfloor \frac{m}{\tau+1} \rfloor$ for the first $k_\tau - |s|\%k_\tau$ partitions and $\lceil \frac{m}{\tau+1} \rceil$ for the other partitions (see the *PlusOne* partition scheme in section 4.1), so that we can easily deduce the location of the substring (including starting and ending position) in D. This procedure is called *query segment instantiation*.
- *Searching against Short Index:* we generate τ-deletion neighborhood for each possible substring which length is within the range $L_{min} - \tau, l_p$ in seg. If the query segment is shorter than l_p, we directly probe it against the index to find all possible match entities. Otherwise, a simplified *query segment instantiation* is needed to find all $2\tau + 1$ possible query segments.

Remarks: The prefix length l_p determines whether a string should be indexed in the short or long inverted list. The larger the prefix length, the more strings indexed in the short inverted list. When l_p is larger than $\frac{L_{max}}{(\tau+1)}$, all strings are only indexed in the short inverted list such that this document processing algorithm degrades into pure deletion neighborhood generation-based algorithm FastSS(Bocek et al. 2007). In real-world application, an optimal value for l_p should be determined empirically.

7 Experimental Study

In this section, PartSS is compared to the two current state-of-the-art methods QGRAM or NGPP when applied to EDJ (Edit Distance similarity Join) or ED-AME (Approximate Membership Extraction with Edit Distance threshold) on several real-world data sets.

- **QGRAM:** The q-gram-based filter which not only contains the three filtering techniques including count, position and length filtering proposed in (Gravano et al. 2001), but also incorporate state-of-art techniques such as prefix filtering (S. Chaudhuri & Kaushik 2006), location-based filering and content-based filtering (Xiao et al. 2008). In the experiments, we set different size of q-grams for strings with different length. The following optimal q-gram size for string s with different length is adopted from the paper (Wang et al. 2009).

$$q = \begin{cases} 2 & (1 \leq |s| \leq 13) \\ 3 & (12 \leq |s| \leq 20) \\ 4 & (|s| \geq 18) \end{cases}$$

- **NGPP:** The Neighborhood Generation with Partitioning and Prefix-based pruning, proposed in (Wang et al. 2009).
- **PartSS:** The Partition-based filtering with Scaling and Shifting operations proposed in this paper.

7.1 Experimental Settings

We evaluate the performance of the three filters in EDJ on the following three real-world datasets.

- **DBLP-Papers:** *A set of 300k randomly selected paper titles from the DBLP database* [1].
- **GENE:** *A random selection of 200k entities from more than 1M Gene/Protein lexicon generated from MEDLINE documents by (Tanabe & Wilbur 2004).*
- **TEXAS:** *150k records from Texas* [2]*, a text dump of Broker and Sales Licensees database from the Texas Real Estate Commission. Texas has been used in (Li*

[1]http://www.informatik.uni-trier.de/ ley/db
[2]http://www.trec.state.tx.us/LicenseeDataDownloads/trecfile.txt

et al. 2007, Xiao et al. 2008). Each record in Texas is a concatenation of 19 attributes, including person names, addresses, and licence information.

We also evaluate the performance of the three filters in ED-AME on the three real-world datasets below.

- **DBLP-Persons:** *A set of 250k person's names randomly selected from the DBLP bibliography dataset as the dictionary. The documents are 5k references extracted from CiteSeer* [3]*, each containing paper id, author, title and venue information.*
- **GENE:** *The dictionary is the GENE we mentioned above. The documents are 10k references from the TREC9 Filtering Track Collections* [4]*. Each reference contains author, title and abstract fields.*
- **CONLL:** *The dictionary is composed of 8.2k entities including person's names, organizations and locations, from the shared task of Conference on Computational Natural Lauguage Learning 2003* [5]*. The documents are 20K news articles from Reuters dataset* [6]*.*

The properties of those datasets are provided in Table 13 and 14 below.

Table 13: Properties of the datasets used for evaluation I

Dataset	Size	Average Length
DBLP-Papers	300k	66.4
GENE	200k	22.4
TEXAS	150k	112.1

Table 14: Properties of the datasets used for evaluation II

Dataset	Size	Average Length
DBLP-Persons (Dict)	250k	13.9
CiteSeer (Doc)	5k	123.2
GENE (Dict)	200k	22.4
TREC (Doc)	10k	1134.5
CONLL (Dict)	8.2k	11.5
Reuters (Doc)	20k	668.5

The edit distance threshold τ is set within range $[1, 3]$, as it covers many important applications (Lee et al. 2007). We also adopt the local edit distance threshold τ' for strings with different length according to (Wang et al. 2009).

$$\tau' = \begin{cases} min(1,\tau) & (1 \leq |s| \leq 5) \\ min(2,\tau) & (6 \leq |s| \leq 11) \\ \tau & (|s| \geq 12) \end{cases}$$

In this section, we mainly compare the performance of the three filters in EDJ and ED-AME through the following two measures: 1) Run_Time: The overall running time, which is the most important factor to judge if a filtering works well; 2) Cand_Size: The numbers of candidate pairs before the verification procedure, which can also reflect the effectiveness of the filtering. Besides, we also measure the size of index entries generated in ED-AME, which reflects the space complexity of filters. Our experimental environment is an Intel 2.13GHz Pentium Dual-Core processor, 2GB memory, running Windows XP Professional. All the approaches are implemented using Java.

[3]http://citeseer.ist.psu.edu/

[4]http://trec.nist.gov/data/t9_filtering.html

[5]http://www.cnts.ua.ac.be/conll2003/ner

[6]http://www.daviddlewis.com/resources/testcollections/reuters21578/

7.2 Measure the Prefix Length

The "prefix length" l_p is the only parameter in our method, hence its optimal value is empirically established through preliminary experiments. The effect of l_p is measured on all datasets by setting $\tau = 1, 2, 3$.

As we can see from the detailed results in Figure 3, as the value of l_p increases (before reaching $\frac{L_{max}}{(\tau+1)}$), the size of index increases, while the size of candidate size decreases. It can be explained by the features of PartSS and FastSS: although the entries of FastSS are more selective than PartSS, FastSS usually generates more entries than PartSS (especially when τ is large). Meanwhile, the overall time to set up the index and do the edit similarity join first decreases, then increases. We conclude that the optimal value of l_p for the GENE is 9 since it reaches the highest efficiency as shown in Figure 3[c]. Through experimental study on other data sets, we found that the optimal value of l_p decreases as the average length of dictionary string entries increases. The optimal value of l_p for other datasets are: $l_p = 6$ for DBLP-Persons, $l_p = 3$ for DBLP-Papers, $l_p = 2$ for TEXAS, $l_p = 8$ for CONLL.

7.3 Edit Similarity Join

We now compare the performance of PartSS and QGRAM filterings in EDJ on three datasets. As we can see in Figure 4(d),(e),(f), the general trend is that running time grows exponentially as the τ increases. For all settings, PartSS is more efficient than QGram. Although QGrams usually prunes even more join pairs than PartSS (Figure 4(a),(b),(c)), the count filtering is still a bottleneck.

7.4 Approximate Membership Extraction

We compare the performance of filters in ED-AME over three datasets. Here the optimal value of l_p for NPGG are: $l_p = 10$ for DBLP-Persons, $l_p = 10$ for GENE, $l_p = 7$ for CONLL. As τ increases, the running time of both methods grows exponentially (Figure 5(g),(h),(i)). For all settings, PartSS is more efficient than NGPP. As expected, the generation of deletion neighborhood for each partition of a query segment by NGPP leads to slower processing, although it shares almost the same searching strategy with PartSS.

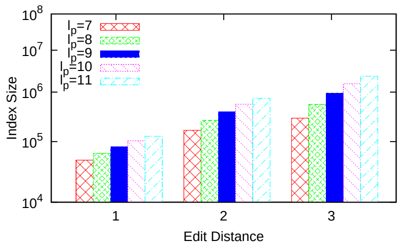
8 Conclusion

In this paper, we proposed a novel partition-based filter PartSS for EDJ and ED-AME, which improves over the state-of-the-art method NGPP with the implementation of a new partitioning scheme and variants generation rules. Without generating *deletion neighborhood*, we need less space and time to filter more false positive string pairs than NGPP. We presented how this filter can be applied into both EDJ and ED-AME and an experimental study based on several real-world data sets demonstrated the effectiveness of the filtration based on our partition-based scheme.

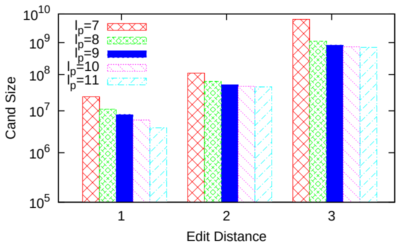
As future works, we will apply this filter into real-world applications such as retrieving results for typo keywords in information retrieval or recognizing homologous DNA strands in bioinformatics. To reach the best performance in our target applications, we will determine how to combine several filter techniques.

References

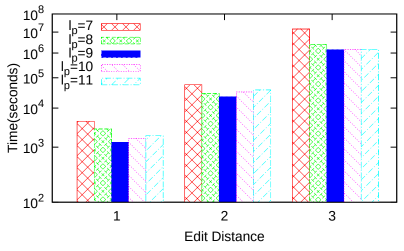
Arasu, A., Ganti, V. & Kaushik, R. (2006), Efficient exact set-similarity joins, *in* 'Proceedings of the 32nd International Conference on Very Large Data Bases', p. 929.



(a) GENE, Index_Size

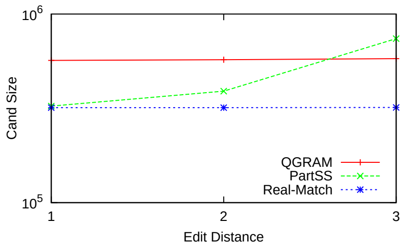


(b) GENE, Cand_Size

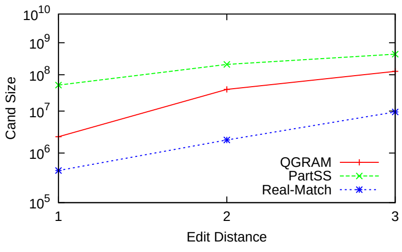


(c) GENE, Time

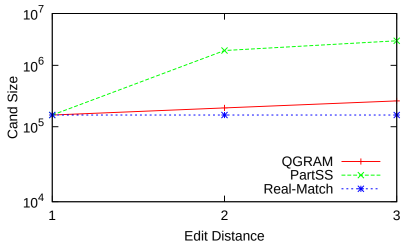
Figure 3: The Effect of Prefix Length on GENE DataSet)



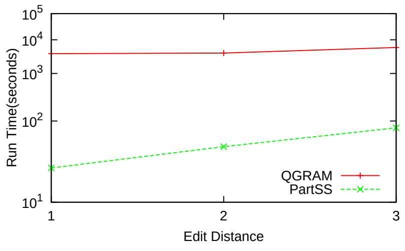
(a) DBLP-Papers, Cand_Size



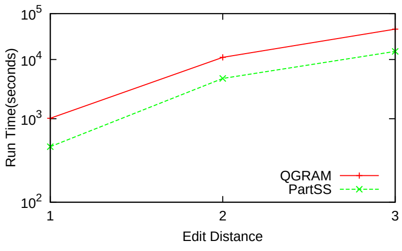
(b) GENE, Cand_Size



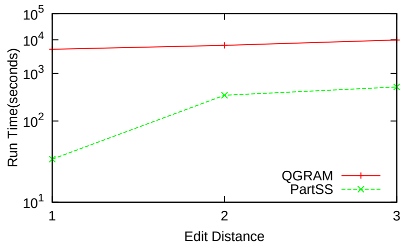
(c) TEXAS, Cand_Size



(d) DBLP-Papers, Run_Time



(e) GENE, Run_Time



(f) TEXAS, Run_Time

Figure 4: Experimental Results for Edit Similarity Join

Bayardo, R., Ma, Y. & Srikant, R. (2007), Scaling up all pairs similarity search, *in* 'Proceedings of the 16th International Conference on World Wide Web', p. 140.

Bilenko, M., Mooney, R., Cohen, W., Ravikumar, P. & Fienberg, S. (2003), 'Adaptive name matching in information integration', *IEEE Intelligent Systems* **18**(5), 16–23.

Bocek, B., Hunt, E. & Stiller, B. (2007), 'Fast Similarity Search in Large Dictionaries'.

Chakrabarti, K., Chaudhuri, S., Ganti, V. & Xin, D. (2008), An efficient filter for approximate membership checking, *in* 'Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data', pp. 805–818.

Chandel, A., Nagesh, P. & Sarawagi, S. (2006), Efficient batch top-k search for dictionary-based entity recognition, *in* 'Proceedings of the 22nd ICDE International Conference on Data Engineering', p. 28.

Gionis, A., Indyk, P. & Motwani, R. (1999), Similarity search in high dimensions via hashing, *in* 'Proceedings of the 25th International Conference on Very Large Data Bases', pp. 518–529.

Gravano, L., Ipeirotis, P., Jagadish, H., Koudas, N., Muthukrishnan, S. & Srivastava, D. (2001), Approximate string joins in a database (almost) for free, *in* 'Proceedings of the 27th International Conference on Very Large Data Bases', pp. 491–500.

Lee, H., Ng, R. & Shim, K. (2007), Extending q-grams to estimate selectivity of string matching with low edit distance, *in* 'Proceedings of the 33rd International Conference on Very Large Data Bases', pp. 195–206.

Li, C., Wang, B. & Yang, X. (2007), VGRAM: Improving performance of approximate queries on string collections using variable-length grams, *in* 'Proceedings of the 33rd International Conference on Very Large Data Bases', pp. 303–314.

Lu, J., Han, J. & Meng, X. (2009), Efficient algorithms for approximate member extraction using signature-based inverted lists, *in* 'Proceedings of the 18th ACM Conference on Information and Knowledge Management', pp. 315–324.

Masek, W. & Paterson, M. (1980), 'A faster algorithm computing string edit distances* 1', *Journal of Computer and System sciences* **20**(1), 18–31.

Navarro, G. (2001), 'A guided tour to approximate string matching', *ACM Computing Surveys (CSUR)* **33**(1), 88.

S. Chaudhuri, V. G. & Kaushik, R. (2006), A Primitive Operator for Similarity Joins in Data Cleaning, *in* 'Proceedings of the 22nd International Conference on Data Engineering'.

Sarawagi, S. & Bhamidipaty, A. (2002), Interactive deduplication using active learning, *in* 'Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining', pp. 269–278.

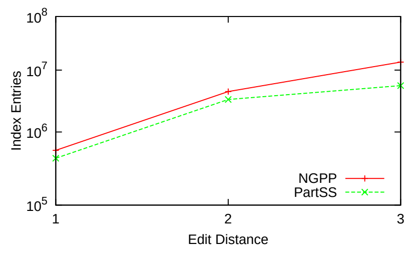
Sarawagi, S. & Kirpal, A. (2004), Efficient set joins on similarity predicates, *in* 'Proceedings of the 30th ACM SIGMOD International Conference on Management of Data', p. 754.

Singhal, A. (2001), 'Modern information retrieval: A brief overview', *IEEE Data Engineering Bulletin* **24**(4), 35–43.

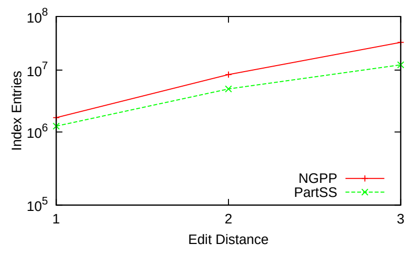
Tanabe, L. & Wilbur, W. (2004), 'Generation of a large gene/protein lexicon by morphological pattern analysis', *Journal of Bioinformatics and Computational Biology* **1**(4), 611–626.

Ukkonen, E. (1983), On approximate string matching, *in* 'Foundations of Computation Theory', pp. 487–495.

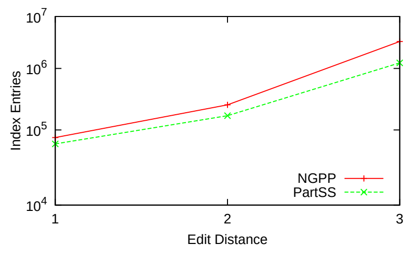
Wagner, R. & Fischer, M. (1974), 'The string-to-string correction problem', *Journal of the ACM (JACM)* **21**(1), 168–173.



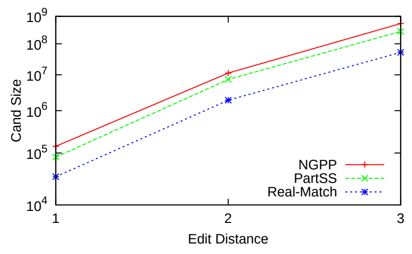
(a) DBLP-Persons, Index_Size



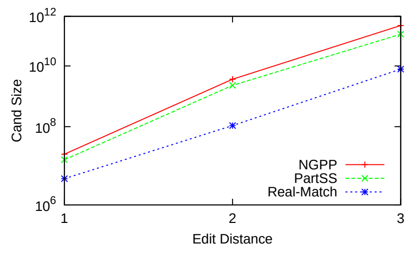
(b) GENE, Index_Size



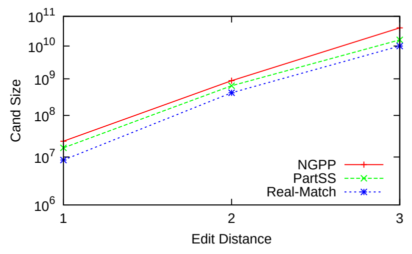
(c) CONLL, Index_Size



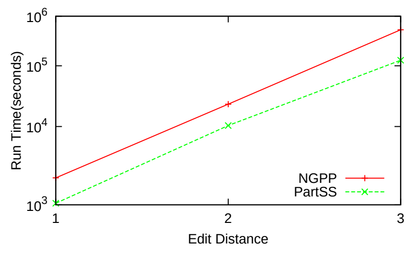
(d) DBLP-Persons, Cand_Size



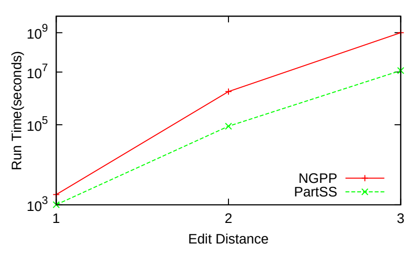
(e) GENE, Cand_Size



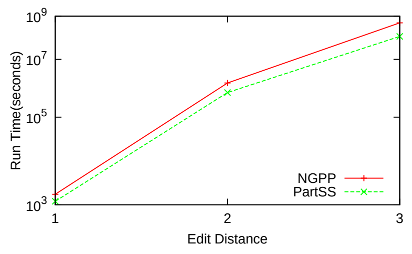
(f) CONLL, Cand_Size



(g) DBLP-Persons, Run_Time



(h) GENE, Run_Time



(i) CONLL, Run_Time

Figure 5: Experimental Results for Approximate Membership Extraction with Edit Distance Constraints

Wang, W., Xiao, C., Lin, X. & Zhang, C. (2009), Efficient approximate entity extraction with edit distance constraints, *in* 'Proceedings of the 35th SIGMOD International Conference on Management of Data', pp. 759–770.

Winkler, W. (1999), 'The state of record linkage and current research problems', *Statistical Research Division, US Bureau of the Census, Wachington, DC* .

Xiao, C., Wang, W. & Lin, X. (2008), 'Ed-Join: an efficient algorithm for similarity joins with edit distance constraints', *Proceedings of the VLDB Endowment* **1**(1), 933–944.

Zobel, J. & Dart, P. (1995), 'Finding approximate matches in large lexicons', *Software: Practice and Experience* **25**(3), 331–345.