A possibilistic approach on string comparison

Antoon Bronselaer, Guy De Tré
Department of Telecommunications and Information Processing, Ghent University
Sint-Pietersnieuwstraat 41, B-9000 Ghent, Belgium

The authors are grateful to Saskia Debergh and Dirk Van Hyfte from i.Know for their input on tokenization, the fruitful discussions and the delivery of the biomed dataset

Abstract

In this paper, comparison of strings is tackled from a possibilistic point of view. Instead of using the concept of similarity between strings, co-reference between strings is adopted. The possibility of co-reference is estimated by means of a possibilistic comparison operator. In literature, two important classes of comparison methods for strings have been distinguished: *character based* methods and *token based* methods. The first class treats a string as a sequence of characters, while the second class treats a string as a vector of substrings. The first contribution of this paper is to propose a new character based method that is able to detect typographical errors and abbreviations. The main advantage of our technique is the very low complexity in comparison with existing character based techniques. In a second contribution, two-level systems are investigated and a new approach is described. The novelty of the proposed two-level system is the use of multiset comparison rather than vector comparison. It is shown how an ordered weighted conjunctive operator that uses a parameterized fuzzy quantifier to deliver weights, is competitive with frequency based weights. In addition, the use of a quantifier is significantly faster than the use of existing weight techniques. In a third contribution, a novel class of hybrid techniques is proposed that combines the advantages of several methods. Finally, comparative tests regarding accuracy and execution time are performed and reported.

Index Terms

String matching, Possibility theory, Algorithms, Operators (mathematics), Fuzzy logic

I. INTRODUCTION

Comparison of strings is a widely studied area, with many useful applications in, amongst others, information retrieval, data cleansing and duplicate detection. In general, two important classes of methods can be distinguished: *character* based methods and *token* based methods. The fundamentals of character based methods are due to Levenstein in 1966 [1], who defined the *edit distance* (or *Levenstein distance*) between two strings s and t as the minimal number of edit operations required to transform s into t, where the allowed operations are character insertion, character deletion and character substitution. Damerau considered *transposition* as an additional edit operation in [2]. He stated that these operations correspond to more than 80% of all human misspellings. Needleman and Wunsch [3] defined a similarity measure for protein sequences based on the longest common subsequence (LCS). Smith and Waterman modified this measure to local alignments in [4] and Monge and Elkan were the first to recognize the importance of the previous works in word comparison in [5], by proposing an affine gap extension of edit distance that copes with abbreviations. Next to these methods, another important character based method is the Jaro measure ([6]), which is a heuristic method that is suitable for short strings and is significantly faster than the methods already mentioned. Winkler extended this method based on the observation that the probability of an error in the beginning of a string is relatively low. A comparative study of these metrics and measures was performered in [7] (using the implementation in [8]). The Monge-Elkan approach showed the best overall accuracy (on average), followed closely by the Jaro measure. The same study also showed that Monge-Elkan is an order of magnitude slower that Jaro and that Monge-Elkan tends not to be very robust. Hence, Jaro seems to combine a relative high accuracy with a relative high performance. This paper adopts the idea of combining high accuracy with high performance. Therefore we start by formalizing some operators on strings that are used in the affine gap strategy. Next, some important properties of these operators are given, leading to the idea that we can provide low complexity approximations of these operators. Applying these approximations results in a very fast operator. The assumptions under which the approximations are equal to the affine gap strategy are given and it is shown that these assumptions are more likely to be violated if longer strings are used. Therefor, the proposed character based method is best used in a two-level system.

Next to the character based methods, token based methods are an important second class of comparison methods. Hereby, a string is first tokenized and the problem translates to vector comparison. An example of such a method is the cosine similarity with TFIDF weighting scheme ([9]). While character based methods are suitable for errors on character level such as typographical errors or abbreviations, token based methods are capable of detecting errors on word level. In addition, the study in [7] showed that token based methods are an order of magnitude faster than the fastest character based method (Jaro). In comparison to the Jaro measure other character based methods are typically an order of magnitude slower. Therefor, to combine the best of two worlds, recent studies proposed the combination of character based and token based methods, leading to two-level systems([5], [10], [11]), where the first step performs comparison on the character level and the second step performs comparison on the word level. The study in [7] recognized SoftTFIDF with Jaro-Winkler as low level similarity measure, as the best method with respect to average accuracy and stated that the execution time of two-level systems depends strongly on the used low level method. Therefor, a two-level system is constructed that uses the novel and fast character based operator for strings as a low level method. Instead of using vector comparison (as with SoftTFIDF), multiset comparison is proposed, consisting of two major steps: an injective mapping and an aggregation operator. An algorithm for the mapping construction has been provided in [12]. As an aggregation operator, the use of ordered weighted conjunction is proposed, where the weight vector is delivered by a parameterized fuzzy quantifier. Using such a quantifier has two major benefits: the construction of the weight vector is much faster than with TFIDF weights and the quantifier approach has higher accuracy than SoftTFIDF on several tested datasets. After construction of the new two-level system, a hybrid approach is investigated that combines both evidence delivered by the character based approach and the two-level approach. It is shown how weighted disjunction can be used as combinatory function.

As a comparative model for strings, the notion of *co-reference* of two strings is preferred over the notion of similarity, due to t-transitivity (where t is a t-norm) as an axiomatical requirement of similarity. Having two strings s and t, a proposition $p_{s,t}$="s and t are co-referent" signifies that s and t describe the same real world entity. Clearly, such relations do not satisfy transitivity at all. Instead of estimating the similarity between s and t, we estimate the uncertainty that is related to the boolean value of $p_{s,t}$ and express this as a possibilistic truth value. Possibilistic truth values are chosen here due to their intuitive modeling of uncertainty. An operator that provides such a possibilistic truth value is called a *possibilistic comparative evaluation operator* or evaluator for short. The concept of evaluators is introduced in [12], where an evaluator for sets is designed. The comparison methods proposed in this paper are all evaluators. More on possibility theory and possibilistic truth values can be found in [13],[14] ,[15] and [16].

The remainder of this paper is structured as follows. In Section II basic notations and definitions on strings are provided and some important properties are given. Based on these properties, a character based evaluator (also called one-level evaluator in the following) for strings is given in Section III. Next, in Section IV a novel two-level system approach is defined and a description of hybrid systems is given. Section V compares the accuracy of the two-level system and the hybrid system with SoftTFIDF and reports on the performance of the introduced methods. Section VI provides a brief overview of future work. Finally, Section VII summarizes the most important contributions of this paper.

II. Strings and n-languages

In this Section, basic operators for string manipulation are formally defined and properties of these operators are investigated. Within the scope of this paper, the term 'string' indicates an instance of a data type to represent words of a natural language. Such a natural language is always based on an alphabet A. The elements of this alphabet A are called characters. On data type level, a string is nothing more than a list of characters:

Definition 1 (String of length n)

A string s of length n drawn from an alphabet A is specified by a mapping function ι_s:

$$\iota_s : I_n \rightarrow A$$

where $I_n = \{1, 2, .., n\}$. A string with length 0 is called an empty string and is denoted by σ. The set of all strings of length n is denoted $\mathcal{S}_n$ and the set of all strings is denoted $\mathcal{S}$.

The length of a string s is denoted $|s|$, which is by definition equal to $|I_n|$. Two strings s and t are equal if both the index sets and the images of ι are equal:

$$s = t \Leftrightarrow I_{|s|} = I_{|t|} \wedge \forall i \in I_{|s|} : \iota_s(i) = \iota_t(i)$$

The transformation operator for strings [.] is formally defined by:

Definition 2 (Transformation operator)

Given a string $s \in \mathcal{S}_n$ with index set I_n. Assume $q = (q_1, ..., q_k)$ with $k \leq n$. For all i and j, $q_i \in I_n \wedge i \neq j \Rightarrow q_i \neq q_j$. The transformation of s under q is a string $s[q] \in \mathcal{S}_k$ with:

$$\forall i \in \{1, 2, ..., k\} : \iota_{s[q]}(i) = \iota_s(q_i)$$

The transformation operator [.] has some interesting special cases. If $q_{i+1} - q_i = 1$, the transformed string is called a substring of s.

Definition 3 (Substring)

Assume two strings $s, t \in \mathcal{S}$, then t is a substring of s, denoted $t \sqsubset s$ if:

$$t = s[q]$$

with $q_{i+1} - q_i = 1$

Weakening the constraint on elements of the transformation vector from monotonic with increase 1 to just monotonic, implies a weakening of the definition of substring:

Definition 4 (Weak substring)

Assume two strings $s, t \in \mathcal{S}$, then t is a weak substring of s, denoted $t \sqsubset_w s$ if:

$$t = s[q]$$

with $q_i < q_{i+1}$

From a theoretical point of view, (weak) substrings of s are projections on the index set of s. Using the term *weak* substring is justified by noting that the definition of a weak substring is a generalization of the definition of substring:

$$s \sqsubset t \Rightarrow s \sqsubset_w t$$

Next, string concatenation is defined.

Definition 5 (String concatenation)

Assume two string $s, t \in \mathcal{S}$. The concatenation of s and t, denoted as $s \oplus t$, is a string $r \in \mathcal{S}_{|s|+|t|}$ with ι_r:

$$\forall i \in I_{|s|+|t|} : \iota_r(i) = \begin{cases} \iota_s(i), & i \leq |s| \\ \iota_t(i - |s|), & i > |s| \end{cases}$$

The following properties are satisfied:

$$(s \oplus t) \oplus r = s \oplus (t \oplus r)$$
$$s \oplus \sigma = s = \sigma \oplus s$$

After introducing the relevant operators for string manipulation, a more general concept is introduced. The definition of an n-language is given, providing an elegant model for reasoning with strings. Formally:

Definition 6 (n-language)

A language of length n, also called an n-language, drawn from an alphabet A is a set of strings s with length n. The set of all languages of length n is denoted $\mathcal{L}_n$ and the set of all languages is denoted $\mathcal{L}$.

Definition 6 implies that any n-language $L \subset \mathcal{S}_n$, which means that sublanguages, language union and language intersection are unambiguously defined in terms of set operations. The definition of n-languages provided here is a special case of formal languages. Defining them has, from our point of view, a major benefit. Reasoning with strings frequently has to cope with uncertainty. Definition 6 offers an elegant framework to model this uncertainty. When dealing with n-languages, two important concepts are cardinality and characteristic length.

Definition 7 (Cardinality and characteristic length)

The cardinality of an n-language L is denoted $|L|$ and is equal to the number of strings in the language, i.e. the set cardinality. The characteristic length of an n-language L is denoted $cl(L)$ and is equal to n.

A language L that contains a string s is denoted $s \in L$. Clearly, a string is a special case of an n-language:

$$\forall s \in \mathcal{S} : s \equiv L \Leftrightarrow L \in \mathcal{L}_n \wedge s \in L \wedge |L| = 1$$

Definition 8 (Language concatenation)

Assume $L \in \mathcal{L}_n$ and $K \in \mathcal{L}_m$. The concatenation of L and K, denoted as $L \oplus K$, is an $(n+m)$-language P specified as:

$$\forall (s,t) \in (L \times K) : (s \oplus t) \in P$$

We use $\oplus$ both to denote string concatenation and language concatenation, which is no problem as concatenation of languages is a generalization of concatenation of strings. It should always be clear from the premises which operator is used. As mentioned before, the main reason to define n-languages is to allow elegant reasoning with strings. Therefore, it is of interest to know the relationships between the strings in an n-language. For that purpose, the concept of character sets is defined.

Definition 9 (Character set)

Given an n-language L, the i^{th} character set of L is the subset of the alphabet containing possible elements of A that occur on the i^{th} index position of the strings in the language. Formally:

$$C_{L,i} = \{c | \exists s \in L : \iota_s(i) = c\}$$

The global character set of L is denoted C_L with:

$$C_L = \bigcup_{i=1}^{cl(L)} C_{L,i}$$

The use of n-languages can be further exploited by using them to define the *intersection* of two strings.

Definition 10 (String intersection)

Assume k strings $s_i \in \mathcal{S}$. The intersection of strings $s_i, i = 1, ..., k$, denoted as $s_1 \sqcap ... \sqcap s_k$, is an n-language L for which:

$$(\forall r \in L : \forall i \in \{1, ..., k\} : r \sqsubset s_i) \wedge (\forall u \in \mathcal{S} \backslash L \wedge |u| \geq n : \exists i \in \{1, ..., k\} : \neg(u \sqsubset s_i))$$

In words, the intersection of two strings s and t is the language containing all longest strings that are substring of both s and t. The rationale to call this an intersection stems from the fact that the intersection of two sets A and B is the largest set (in terms of set cardinality) that is a subset of both A and B. The same principle is also used by Zadeh to define the generalization of $\cap$ for fuzzy sets in [17]. From Definition 10, it can be seen that the following properties are satisfied:

$$s \sqcap t = t \sqcap s \qquad \text{(Commutativity)}$$

$$s \sqcap s = \{s\} \qquad \text{(Idempotency)}$$

$$s \sqcap \sigma = \{\sigma\} \qquad \text{(Absorbing element)}$$

$$cl(s \sqcap t) \leq \min(|s|, |t|) \qquad \text{(n-limit)}$$

Definition 10 implies also the definition of s-intersection of languages (the term s-intersection is used because language intersection is actually set intersection). Having $k + 1$ strings, the intersection of the k first strings is an n-language, say L, by definition. Also, the intersection of all $k + 1$ strings is an n-language, say K, which means that the intersection of L with the $(k + 1)^{st}$ string is unambiguously defined. From this point of view, it makes sense to state that:

$$(s \sqcap t) \sqcap r = s \sqcap (t \sqcap r) \qquad \text{(Associativity)}$$

$$s_1 \sqcap ... \sqcap s_{k+1} \in \mathcal{L}_{n \leq cl(s_1 \sqcap ... \sqcap s_k)} \qquad \text{(n-monoticity)}$$

Based on intersection of two strings, it is possible to define string difference formally:

Definition 11 (String difference)

Assume two strings $s, t \in \mathcal{S}$. The string difference $s \ominus t$, is an $(|s| - cl(s \sqcap t))$-language L with $|L| = |s \sqcap t|$ for which:

$$\forall r \in L : \exists u \in (s \sqcap t) : u = s[q] \wedge r = s[q']$$

with q' the complementary index vector of q with respect to $(1, 2, ..., |s|)$ and $q'_i < q'_{i+1}$.

Clearly, string difference does not satisfy the property of commutativity, which is why n-ary string difference is not further investigated here. The concepts introduced so far can be weakened by replacing $\sqsubset$ with $\sqsubset_w$, which leads to the following important definitions:

Definition 12 (Weak string intersection)

Assume k strings $s_i \in \mathcal{S}$. The weak intersection of strings $s_i, i = 1, ..., k$, denoted as $s_1 \sqcap_w ... \sqcap_w s_k$, is an n-language L for which:

$$(\forall r \in L : \forall i \in \{1, ..., k\} : r \sqsubset_w s_i) \wedge (\forall u \in \mathcal{S} \backslash L \wedge |u| \geq n : \exists i \in \{1, ..., k\} : \neg(u \sqsubset_w s_i))$$

As the concept of weak substring is a generalization of the concept of substring, all remarks on string intersection so far, are valid for weak string intersection. In addition, weak intersection has two important properties that are not satisfied for regular string intersection. These properties are left and right distributivity of $\oplus$ over $\sqcap_w$:

$$t \oplus (s_1 \sqcap_w ... \sqcap_w s_k) = (t \oplus s_1) \sqcap_w ... \sqcap_w (t \oplus s_k) \qquad \text{(Left distributivity of } \oplus \text{ over } \sqcap_w)$$

$$(s_1 \sqcap_w ... \sqcap_w s_k) \oplus t = (s_1 \oplus t) \sqcap_w ... \sqcap_w (s_k \oplus t) \qquad \text{(Right distributivity of } \oplus \text{ over } \sqcap_w)$$

Definition 13 (Weak string difference)

Assume two strings $s, t \in \mathcal{S}$. The weak string difference $s \ominus_w t$, is an $(|s| - cl(s \sqcap_w t))$-language L with $|L| = |s \sqcap t|$ for which:

$$\forall r \in L : \exists u \in (s \sqcap_w t) : u = s[q] \wedge r = s[q']$$

with q' the complementary index vector of q with respect to $(1, 2, ..., |s|)$ and $q'_i < q'_{i+1}$.

Weak intersection has important properties with respect to the edit operations defined by Levenstein and Damerau. Assume strings $s, t, a, b \in \mathcal{S}$ and $c, d \in \mathcal{S}_1$, then the following properties are satisfied:

$$(s = a \oplus c \oplus b) \wedge (t = a \oplus b) \Rightarrow (s \sqcap_w t = \{a \oplus b\})$$

$$(s = a \oplus c \oplus d \oplus b) \wedge (t = a \oplus d \oplus c \oplus b) \Rightarrow (s \sqcap_w t = \{a \oplus c \oplus b, a \oplus d \oplus b\})$$

which are corollaries of the distributive laws of $\oplus$ over $\sqcap_w$. If the intersection of two strings is an n-language with $n-1$ singleton character sets, the n-language is unambiguously determined as the cross product of the character sets. Formally, assume without loss of generalization that the first $n-1$ character sets are singleton sets:

$$\forall i \in \{1, ..., cl(s \sqcap_w t) - 1\} : |C_{s \sqcap_w t, i}| = 1 \Rightarrow s \sqcap_w t = C_{s \sqcap_w t, 1} \times ... \times C_{s \sqcap_w t, n}$$

It follows that in this case, we can put a moving window over the strings to construct the intersection. More specific, we start at the beginning of both strings and move a window over them. While doing so, common characters under the window in both strings are added to the intersection. For example, assume s="john B" and t="jon B". The construction of $s \sqcap_w t$ is illustrated in Figure II. In step (a) and (b), the window indicates respectively characters 'j' and

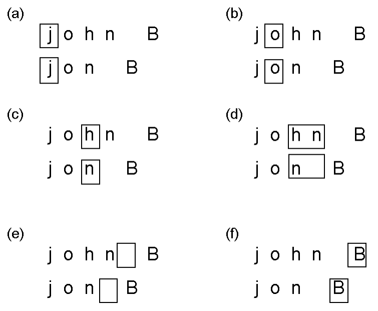

Fig. 1. Linear construction of weak intersection

'o' in both strings. Hence, both characters are added to the intersection. In step (c), the window indicates substrings in s and t that do not have any character in common. Hence, the window size is increased with 1. In step (d), the window indicates substrings in s and t that have one character ('n') in common, which is added to the intersection. This causes the window to shrink and move in step (e), so that the window indicates in both strings the substring ' ', which is added to the intersection. Finally, in step (f) character 'B' is added to the intersection. This construction leads to $s \sqcap_w t$ = {"jon B"}, which satisfies the definition of weak intersection.

It is emphasized that without the above mentioned assumption, the intersection constructed in such a moving window approach can be incorrect. The left and right distributivity of $\oplus$ over $\sqcap_w$ can be used to determine the cases in which such an approach provides the correct solution. An example of such cases are the corollaries of the distributivity laws with respect to edit operations. Further on, assume three strings s, t and r, then we have:

$$(C_r \cap (C_s \cup C_t)) = \emptyset \Rightarrow (s \oplus t) \sqcap_w (s \oplus r \oplus t) = \{s \oplus t\}$$

which implies that in this case the linear approximation provides the correct result.

III. One-level string comparison

In this section, a new one-level comparison model for strings is discussed, based on the operators and their properties discussed in the previous Section. Unlike all existing techniques, our model does not adopt the notion of similarity or distance to express results. Instead, we measure *possibility of object co-reference*. Co-referent objects are descriptions of a real world entity that are not necessarily equal, but describe the same entity. The difference between co-reference and similarity is subtle, but existent. Theoretically, a similarity relation satisfies reflexivity, symmetry and t-transitivity (with t an arbitrary t-norm) and is therefor a t-transitive proximity relation ([18]). However, in recent work, it is shown that the co-reference relation does not satisfy transitivity [12]. This work defines *evaluators* as a machinery to measure the possibility that two arbitrary objects are (not) co-referent. More specific, for arbitrary objects a and b, it assumes p to be an affirmative proposition that states ‘a and b are co-referent’ and measures the possibility that p evaluates to T (true) and the possibility that p evaluates to F (false). The result of such an evaluator is thus a possibilistic truth value, reflecting uncertainty about the co-reference of two objects. Such an evaluator can be defined in case the objects are strings:

Definition 14 (One-level string evaluator)
Given the set of all strings $\mathcal{S}$, for each couple of strings $(s,t) \in \mathcal{S}^2$, an affirmative proposition can be stated:

p = “s and t are co-referent”

The knowledge about this proposition is given by the evaluator function:

$$E_{\mathcal{S}} : \mathcal{S}^2 \rightarrow \tilde{\wp}(I) : (s,t) \mapsto E_{\mathcal{S}}(s,t)$$

where $I = \{T, F\}$ and $\tilde{\wp}(I)$ is the set of all fuzzy sets that can be defined over I. Further on:

$$E_{\mathcal{S}}(s,t) = \tilde{p} = \{(T, \mu_{\tilde{p}}(T)), (F, \mu_{\tilde{p}}(F))\}$$

where $\mu_{\tilde{p}}(T)$ is the possibility that p is true and $\mu_{\tilde{p}}(F)$ is the possibility that p is false and $E_{\mathcal{S}}$ must satisfy:

$$E_{\mathcal{S}}(s,t) = E_{\mathcal{S}}(t,s)$$

We will formulate the measurement of the possibilities in the following, thereby explicitly stating that abbreviations and misspellings can cause non-equality of co-referent descriptions. For example: “Main Street” and “Main Str” are not equal, but the uncertainty about their co-reference is rather low. Unless stated otherwise, it is assumed in the following that evaluators are *strong reflexive*:

$$E_{\mathcal{S}}(s,t) = \{(T,1)\} \Leftrightarrow s = t$$

A. Estimating $E_{\mathcal{S}}(s,t)$

From a study of literature, we found that Monge and Elkan have defined a metric for strings (in the context of natural languages) that adopts the idea of evaluating both the longest common subsequence and the differences between the strings. More specific, they are able to recognize abbreviations. As mentioned before, studies have shown that this metric is actually rather slow in execution time. More specific, the dynamical programming algorithm to compute the Monge-Elkan distance has quadratic complexity. In addition, the algorithm uses a similarity measure on character level (instead of character equality) which increases the complexity. Therefor, a new one-level comparison method that satisfies Definition 14 is proposed that allows to take into account misspellings and abbreviations, but has low complexity. The estimation of the possibilities is based on an approximation of weak intersection, which is mentioned briefly in the previous Section. The method puts a moving window over both strings at the same time and computes the possibilities increasingly. For each string, a buffer is used, say bs and bt for strings s and t. These buffers are filled up as the window advances. As long as the character

Algorithm 1 String Evaluator

```
Require: $(s,t) \in \mathcal{S}^2$
Ensure: $(\mu_{\tilde{p}}(T), \mu_{\tilde{p}}(F))$ represents *co-referential uncertainty* of $s$ and $t$
1: $(next, ps, pt, w, flag1, flag2) \leftarrow (\textbf{true}, 0, 0, 0, \textbf{false}, \textbf{false})$
2: repeat
3:     if $ps + w \leq |s|$ then
4:         $a \leftarrow \iota_s(ps + w)$
5:     else
6:         $(flag1, flag2) \leftarrow (\textbf{true}, flag2 \vee (ps = |s|))$
7:     end if
8:     if $pt + w \leq |t|$ then
9:         $b \leftarrow \iota_t(pt + w)$
10:    else
11:        $(flag1, flag2) \leftarrow (flag1 \vee (pt = |t|), \textbf{true})$
12:    end if
13:    if $flag1 \wedge flag2$ then
14:        $(next, y) \leftarrow (\textbf{false}, y + \textbf{cost}(|s| - ps + w, |t| - pt + w))$
15:    else
16:        $(sind, tind) \leftarrow (\textbf{index}(s[ps, ..., \min(|s|, ps + w)], b), \textbf{index}(t[pt, ..., \min(|t|, pt + w)], a)$
17:        if $sind = -1 \wedge tind = -1$ then
18:            $w \leftarrow w + 1$
19:        else if $(\min(sind, tind) > -1 \wedge sind < tind) \vee tind = -1$ then
20:            $(x, y, ps, pt, w) \leftarrow (x + 1, y + \textbf{cost}(sind, w), sind + 1, w + 1, 0)$
21:        else if $(\min(sind, tind) > -1 \wedge tind < sind) \vee sind = -1$ then
22:            $(x, y, ps, pt, w) \leftarrow (x + 1, y + \textbf{cost}(w, tind), w + 1, tind + 1, 0)$
23:        else
24:            $(x, y, ps, pt, w) \leftarrow (x + 1, y + \textbf{cost}(w, w), w + 1, w + 1, 0)$
25:        end if
26:    end if
27: until $next = false$
28: $x \leftarrow \frac{x}{\max(|s|,|t|)}$
29: $(\mu_{\tilde{p}}(T), \mu_{\tilde{p}}(F)) \leftarrow (\frac{x}{\max(x,y)}, \frac{y}{\max(x,y)})$
```

sets of the buffers are disjunct, the character indicated by the window size and a pointer is concatenated to the buffers. As soon as the buffers share one character c, the algorithm adds c to the approximate weak intersection. Assume that c has index $i_s > 1$ in bs and index $i_t > 1$ in bt, then $bs[(1, ..., i_s - 1)]$ is added to $s \ominus_{w,a} t$ and $bt[(1, ..., i_t - 1)]$ to $t \ominus_{w,a} s$, where a signifies that we construct an approximation. If $i_s = 1$ nothing is added to the difference. The same is true for i_t. Instead of constructing the approximation completely, we directly calculate the required estimations. Algorithm 1 shows our method. The sub procedure **index**(s, c), with

$s \in \mathcal{S}, c \in A$ gives the smallest $i \in I_s$ for which $\iota_s(i) = c$. If no such index can be found, the result of **index**(s, c) is -1. Sub procedure **cost** is responsible for assigning a cost based on the length of the prefixes of the shared character in the buffers. In x, the characteristic length of the approximated weak intersection is stored, while y holds the accumulated cost for differences found during windowing over the strings. Due to the windowing process, Algorithm 1 has low complexity at the cost of having an approximation of the intersection and differences. Clearly,

Algorithm 2 Cost

```
Require: $a, b, |s|, |t| \in I\!N \wedge b_1, b_2$ boundary characters
Ensure: $c$ represents cost
 1: if $a + b = 0$ then
 2:     $c \leftarrow 0$
 3: else if $a \cdot b = 0$ then
 4:     if $\max(a, b) = 1$ then
 5:         $c \leftarrow \frac{1}{f(|s|,|t|)}$
 6:     else
 7:         if $b_1 \in V \vee b_2 \in V$ then
 8:             $c \leftarrow \frac{p \cdot \max(a,b)}{f(|s|,|t|) \cdot \lambda}$
 9:         else
10:             $c \leftarrow \frac{\max(a,b)}{f(|s|,|t|)\lambda}$
11:         end if
12:     end if
13:     $\lambda \leftarrow \frac{\lambda}{cooling}$
14: else
15:     $c \leftarrow \frac{\max(a,b)}{f(|s|,|t|)}$
16: end if
```

the construction of the cost model is of crucial importance to the accuracy of the technique. Algorithm 2 provides a possible cost calculation method, which is very simple and will be used for testing. The model uses a parameter vector $(\lambda, cooling, p)$ where λ is a factor that lowers the cost of finding a gap and $cooling$ lowers λ as more gaps are found. The term 'gap' signifies two characters that have subsequent indexes in one string but not in the other string, and that are subsequently added to the intersection. The gap size is defined as the absolute difference between the two non-subsequent indexes minus 1. For example, consider s="Main Street" and t="Mn Street", then 'M' and 'n' are subsequently added to the intersection. They have subsequent indexes in t (1 and 2), but not in s (1 and 4). The gap size is equal to $|1 - 4| - 1 = 2$. Although the values of λ are only restricted by the condition $\lambda > 0$, we consider the strong constraint that $\lambda \in \{1, 2\}$, which implies that either the gap cost is seen as a regular error (in case abbreviations are not tolerated) or the gap cost increases logarithmic in function of the gap size. Parameter $cooling \geq 1$ reflects the rate at which λ decreases, thereby reflecting the impact of the number of gaps on the possibility that the strings are not co-referent. Concerning the detection of abbreviations, literature considers the existence of a gap to be a satisfying criterion. However, the syntactic structure of words seems to satisfy more stringent criteria. Therefore, the *boundary rule* is introduced. Informally, this heuristic rule indicates whether a gap is actually an abbreviation by checking the *boundary characters* of the gap, which are the characters directly left and right of the gap. For example, assume s="Main Street" and t="Mn Street", then 'a gap between "M" and "n" is present. The boundary characters of this gap are 'M' and 'n'. It is noted that when a gap occurs at the beginning or the end of a string, one of both boundary characters is the null character ϵ. The basic boundary rule proposed here states that if both boundary characters are not equal to ϵ and one of them is a vowel, then the possibility that the gap is caused by an abbreviation is low. This rule is implemented by use of a boundary penalty $p > 1$ as follows. If the antecedent of the rule is satisfied, then $\lambda = \frac{\lambda}{p}$, i.e., the cost assigned to this part of the string difference is increased, reflecting the belief that this gap is not caused by an abbreviation.

B. *Complexity analysis*

A major benefit of the approximated weak intersection is that it is fast in comparison with existing character based methods. To prove this statement, a detailed complexity analysis is performed. Looking at Algorithm 1, there are two major factors that determine the complexity of the algorithm: (i) the number of iterations and (ii) the number of character comparisons to be performed. The number of iterations is studied first. In our analysis it is assumed, without loss of generalization, that $|s| \leq |t|$, to simplify our notations. In the best case, s is a prefix of t and $|s|$ iterations are required, where each iteration increases ps and pt by one. In the worst case, the number of iterations is $O(|s| + |t|)$ because in each iteration either w increases with one or one of the pointers is incremented with $w + 1$. However, if the window for $|t|$ reaches $|t| - 1$, it is possible that the window size shrinks back to zero with ps increased with one. Then, in addition $|s| - 1$ iterations are required in the worst case to rewindow over s. By conclusion, the number of loops or iterations is linear in function of the string lengths. However, an important issue is the complexity of each iteration, which depends on $|s|$ and $|t|$ only in the calculation of $sind$

and $tind$ (line 16 of Algorithm 1). If in iteration i both $sind$ and $tind$ equal -1, the window size increases with one, while the pointers ps and pt remain the same. Consequently, both windows in iteration $i+1$ are one character larger than in iteration i. Note that both windows have always the same size w. The window indicates substrings in s and t, say bs and bt (see line 16 of Algorithm 1), with the following property:

$$C_{bs[(1,\ldots,bs-1)]} \cap C_{bt[(1,\ldots,bt-1)]} = \emptyset$$

Hence, in each iteration, the algorithm takes the last character of bs and bt, resp. variables a and b in the algorithm, and searches a in bt and b in bs. Consequently, the number of character comparisons required in iteration i can be expressed in terms of the window size w. If $w < |s|$ then:

$$\text{comp}_w = \begin{cases} 1 & w = 0 \\ 2(w+1) & w > 0 \end{cases}$$

Else, $|s| \leq w < |t|$, then the number of comparisons in an iteration is $|s| - ps - 1$. Through consecutive iterations, the window size can grow linearly and the bigger the window size, the more comparisons need to be done. Based on the above formula, in case $w < |s|$, the number of character comparisons performed since the last time the window size was reset to 0, is given by:

$$\sum_{i=0}^{w} comp_i = 1 + 2\sum_{i=1}^{w} i + 1 = 1 + \frac{w(w+1)}{2} + 2w = w^2 + 3w + 1$$

from which it follows that the number of comparisons needed to find a new intersection character, is $O(w^2)$. Hence, the strength of the algorithm lies in the fact that a small window size implies lower complexity. The observation that smaller window sizes are more frequently is supported by a test performed on the streets dataset. Figure 2 shows the frequency of the ratios of maximum window size within one comparison over the $\max(|s|, |t|)$, rounded up to 5%. From this illustrative test on a representative dataset (1458*585=852930 comparisons) it can be concluded that the average maximum window size is about 30% of the length of the longer string. Based on these observations, the conclusion is that the presented technique tries to identify several types of string differences with respect to string co-reference, while preserving low complexity. In Section V, performance and accuracy tests are reported that support this conclusion. In the

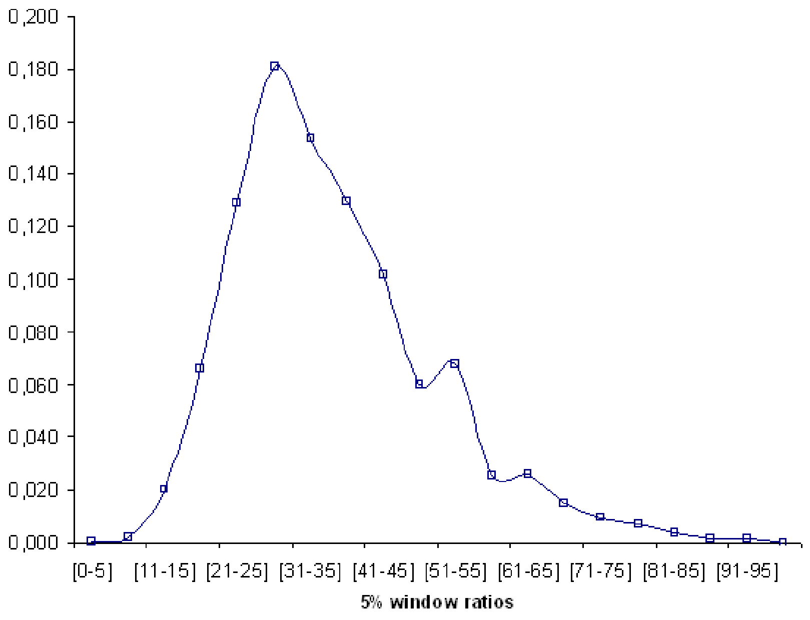

Fig. 2. Frequencies of max. window ratios in streets-dataset

following, the presented technique will be deployed in a pure possibilistic two-level system.

IV. Two-level string comparison

After discussing character based methods to detect string co-reference in the previous Section, this Section focuses on the definition and use of two-level systems. These systems use a *tokenization step* to transform a string s into a multiset of substrings, which are in most cases separate words. The usefulness of these systems is caused by the fact that character based methods are typically not well suited for longer strings. Comparing multisets of substrings instead of large strings has been proven to be a better strategy. The elements of the multisets are compared with a low level method. Hence, both string comparison (first level) and multiset comparison (second level) are performed, which explains the name two-level systems. Such an approach has the clear advantage over character based approaches that substrings are not obliged to be in a specific order. The study of systems where the lower level technique is not based on the equality of strings, is relatively new. The first of these methods are described in [10] and [11], both extending the TFIDF method and thus performing vector comparison, rather than multiset comparison. According to the TFIDF weighting scheme, the weight of a string $a \sqsubset s$ is given by:

$$w(a) = \log(tf_a + 1) \cdot \log(idf_a)$$

with tf_a the number of times a occurs in s, $idf_a = \frac{|DB|}{|n_a|}$, $|DB|$ the size of the database and $n_a = |\{t \in DB | a \sqsubset t\}|$. The idea behind idf_a is that tokens that occur often, have lower weight. For example, the token "Mr." in a database containing employee names can occur often, and is therefor less informative.

The current section is devoted to an extensive study of the class of two-level methods. We begin with formally defining a *tokenization function*:

Definition 15 (Tokenization function $\mathcal{T}_D$)

Given an alphabet A, an alphabet subset $D \subset A$ called the delimiter set and a string s drawn from A. The tokenization function $\mathcal{T}_D$ maps s to the smallest multiset of substrings of s that do not contain a delimiting character. Formally:

$$\mathcal{T}_D : \mathcal{S} \rightarrow \mathcal{M}(\mathcal{S}) : s \mapsto \mathcal{T}_D(s)$$

where

$$\mathcal{T}_D(s) = \min_{|X|}\{X \in \mathcal{M}(\mathcal{S}) | \forall x \in X : x \sqsubset s \wedge \forall i \in I_x : \iota_x(i) \notin D\}$$

Definition 15 implies that tokenization of a string is equivalent to deleting all delimiting characters in s, which can be done in linear time. An important issue is the problem of under tokenization, where to few substrings are produced due to the fact that a delimiter simply can't be found. For example: s="Elm Street" and t="Elmstreet". Such a problem can be solved by concatenating two substrings and verify whether this concatenated string is more useful during comparison. More specific, having two strings s and t and their tokenizations $\mathcal{T}_D(s)$ and $\mathcal{T}_D(t)$, with $a, b \in \mathcal{T}_D(s)$ (assume $a \neq b$) and $c \in \mathcal{T}_D(t)$, if $E_\mathcal{S}(a \oplus b, c) \tilde{>} \tilde{thr}$ with $\tilde{thr}$ a possibilistic truth value serving as threshold, a and b can be removed from $\mathcal{T}_D(s)$ and $a \oplus b$ can be inserted. In addition it is required that $a \oplus x \oplus b \sqsubset s$ with $C_x \subset D$. Checking for all combinations (in both directions) has complexity $O\left(|\mathcal{T}_D(s)||\mathcal{T}_D(t)| - |\mathcal{T}_D(s)| - |\mathcal{T}_D(t)|\right)$, which is quadratic. Given the tokenization function $\mathcal{T}_D$, it is possible to formally define a two-level string evaluator.

Definition 16 (Two-level string evaluator)

Given an alphabet A and the corresponding universe of strings $\mathcal{S}$. A two-level string evaluator in

$\mathcal{S}$, with respect to the delimiting set $D \subset A$ is defined formally as:

$$E^*_{\mathcal{S}} : \mathcal{S}^2 \rightarrow \tilde{\wp}(I) : (s,t) \mapsto E^*_{\mathcal{S}}(s,t)$$

where

$$E^*_{\mathcal{S}}(s,t) = E_{\mathcal{M}(\mathcal{S})}(\mathcal{T}_D(s), \mathcal{T}_D(t))$$

As mentioned before, the main advantage of two-level approaches is the high accuracy on longer strings, relative to the accuracy of character based methods. For an extensive study of (multi)set co-reference, we refer to previous work ([12]). However, a short overview is given in order to understand the rationale. The evaluators for (multi)sets can be classified in two main classes: *hard* evaluators and *soft* evaluators. Hard evaluators evaluate derived (multi)sets such as intersection, symmetrical difference and union, thereby using fuzzy measures to model preferences on element level. As hard evaluators are based on element equality, soft evaluators consider co-reference of elements, thereby using an evaluator on element level. Here, the elements are strings, so this evaluator is $E_{\mathcal{S}}$. The comparison is a two step process. In the first step, an injective element mapping ι is constructed. Hereby, elements of the smallest multiset are mapped to elements of the largest multiset. This mapping implies a sequence of possibilistic truth values, generated by using $E_{\mathcal{S}}$. The second step aggregates the generated PTVs to a final result that states the uncertainty about the co-reference of both (multi)sets. Employing hard evaluators results in systems like WHIRL [9]. Soft evaluators on the other hand result in so called pure two-level systems and therefore, in the remainder of this paper we focus on soft evaluators. The first step (mapping) is done by an algorithm which does not require any modification for use in the current problem ([12]). In the second step, the aggregation operator used is left unspecified. Clearly, a conjunctive operator is needed as it is required in general that a sufficient number of elements occur in both multisets to support the decision of multiset co-reference. The framework of PTVs provides a generalization of Boolean conjunction:

$$\tilde{\wedge} : \tilde{\wp}(I)^2 \rightarrow \tilde{\wp}(I) : \tilde{p}\tilde{\wedge}\tilde{q} \mapsto \{(T, t\left(\mu_{\tilde{p}}(T), \mu_{\tilde{q}}(T)\right)), (F, u(\mu_{\tilde{p}}(F), \mu_{\tilde{q}}(F)))\}$$

where t and u are a pair of t-norm/t-conorm. More on the allowed pairs of norms in a given situation can be found in [16]. Within the context of comparing multisets of strings, such a regular conjunction is clearly to strict. This is caused by the fact that cardinality of co-referent multisets can easily differ, implying the presence of $\{(F,1)\}$ in the generated sequence due to the injective mapping ι. As a consequence, using $\tilde{\wedge}$ means that co-reference of two multisets with a different cardinality is always $\{(F,1)\}$. In [19], a method to model the impact of $[0,1]$-valued weights is given and a definition of (ordered) weighted conjunction is provided. To briefly summarize this work, a function $\mathcal{T}_c$ models the impact of a $[0,1]$-valued weight w as a linear PTV-transformation. Hence:

$$\mathcal{T}_c : [0,1] \times \tilde{\wp}(I) \rightarrow \tilde{\wp}(I) : (w, \tilde{p}) \mapsto \mathcal{T}_c(w, \tilde{p})$$

Using this transformation, extensions of $\tilde{\wedge}$ can be provided. A first approach would be to apply $\tilde{\wedge}$ on transformed PTVs. However, a serious problem is then the determination of the weights. The TFIDF approach is *probabilistic* in nature as it derives weights from frequencies, while the class of two-level systems introduced here is *possibilistic*. It can be illustrated with a simple example why such an initial possibilistic approach will not work. Assume strings s="Mr John Lennon" and t="Lennon John" and the delimiting set containing only the whitespace character. The constructed injection is then as follows:

$$\iota(\text{“John”}) = \text{“John”}$$
$$\iota(\text{“Lennon”}) = \text{“Lennon”}$$

which leads to the following sequence of PTVs:

$$(\{(T,1)\}, \{(T,1)\}, \{(F,1)\})$$

The term frequencies are all 1. Now assume that $|D| = 100$, $n_{\text{“Mr”}} = 30$, $n_{\text{“John”}} = 10$ and $n_{\text{“Lennon”}} = 2$. These assumptions lead to the following normalized weights:

$$w_{\text{“John”}} = 0.589$$
$$w_{\text{“Lennon”}} = 1$$
$$w_{\text{“Mr”}} = 0.308$$

Applying these weights only changes the last PTV in the sequence to $\{(T,1),(F,0.616)\}$. If it is assumed that the propositions are combined by $\tilde{\wedge}$ based on $(\min, \max)$, then the conjunction of the transformed PTVs is equal to $\{(T,1),(F,0.616)\}$. As a result this method indicates that it is quite uncertain whether s and t are co-referent, while it seems obvious they are. Therefore, a second pure possibilistic approach is proposed by using an *ordered weighted conjunction* (OWC). The ordering of PTVs is done by generalized order relations:

$$\tilde{p}_1\ \tilde{R}\ \tilde{p}_2 \Leftrightarrow \begin{cases} \mu_{\tilde{p}_2}(F)\ R\ \mu_{\tilde{p}_1}(F), & \mu_{\tilde{p}_2}(T) = \mu_{\tilde{p}_1}(T) = 1 \\ \mu_{\tilde{p}_1}(T)\ R\ \mu_{\tilde{p}_2}(T), & \text{otherwise} \end{cases}$$

so that a formal definition of ordered weighted conjunction is given by:

Definition 17 (Ordered weighted conjunction)

Given a vector $\underline{\tilde{p}}$ *of PTVs and a vector* $\underline{w}$ *of* $[0,1]$*-values weights with* $\max_i w_i = 1$ *and* $|\underline{\tilde{p}}| = |\underline{w}|$*, the ordered weighted conjunction is given by:*

$$\tilde{\bigwedge}_{ow} : ([0,1])^n \times (\tilde{\wp}(I))^n \rightarrow \tilde{\wp}(I) : (\underline{w}, \underline{\tilde{p}}) \mapsto \tilde{\bigwedge}_{ow}(\underline{w}, \underline{\tilde{p}})$$

where

$$\tilde{\bigwedge}_{ow}(\underline{w}, \underline{\tilde{p}}) = \mathcal{T}_c(w_1, \tilde{p}^s_1)\tilde{\wedge}...\tilde{\wedge}\mathcal{T}_c(w_n, \tilde{p}^s_n)$$

Hereby, $\underline{\tilde{p}}^s$ *is a vector containing all elements of* $\underline{\tilde{p}}$ *but with*

$$\forall i,j \in \{1,...,|\underline{\tilde{p}}|\} : i < j \Rightarrow \tilde{p}^s_i \tilde{\geq} \tilde{p}^s_j$$

The weight vector $\underline{w}$ used by such an OWC can be derived from a parameterized *fuzzy quantifier* that represents the required quantity of co-referent elements in order to decide that the both multisets are co-referent. There is an important semantical difference between TFIDF weights and the weights used here. Instead of calculating the importance of each word (i.e. a substring), the quantifier provides for each quantity of co-referent words, the relevance of this quantity with respect to the decision of multiset co-reference. Clearly, this quantity should be expressed relative to the multiset cardinalities. Therefor, if we assume two multisets X and Y, with $|X| \leq |Y|$, the following quantifier q is proposed:

$$q_{\alpha,\beta,l}(x,|X|,|Y|) = \begin{cases} 1, & x < \alpha|X| \\ 1 + \frac{(l-1)(x-\alpha|X|)}{(1-\alpha-\beta)|X|+\beta|Y|}, & \alpha|X| \leq x \leq |X| + \beta(|Y|-|X|) \\ l, & x > |X| + \beta\left(|Y|-|X|\right) \end{cases}$$

Vector $\underline{w}$ is then derived from q as follows:

$$\forall i \in \{1, .., |Y|\} : w_i = q_{\alpha,\beta,l}(i, |X|, |Y|)$$

Figure IV shows a visual representation of the proposed quantifier q in general and the special case where $\alpha = 1$ and $\beta = 0$. The fuzzy quantifier q will be interpreted here as a possibility

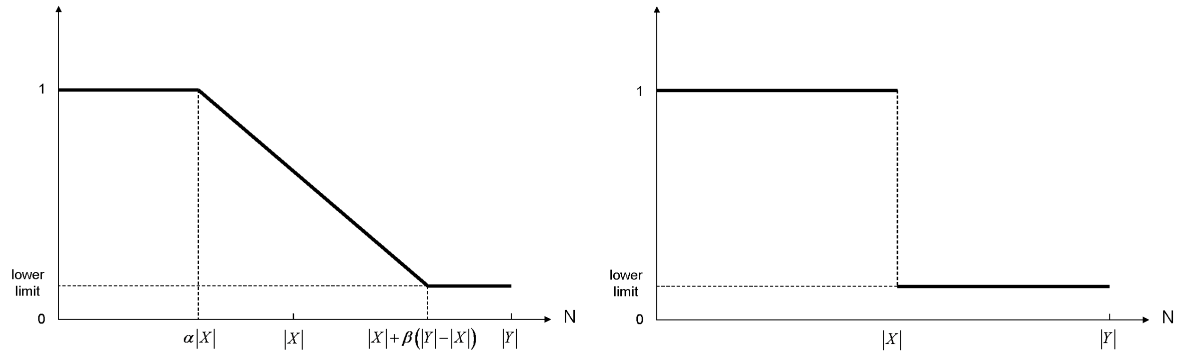

Fig. 3. Parameterized fuzzy quantifiers: vague (left panel) and sharp (right panel)

distribution over $I\!N$. Parameter α expresses the relative portion of co-reference elements that are minimally required in order to conclude that the multisets are co-referent. In case the multiset cardinalities are not equal, parameter β specifies the portion of elements (relative to the difference in cardinality) that have a minimal impact on the decision of multiset co-reference. The proportion of this impact is determined by parameter l, which is typically chosen close to 0. It should be emphasized that if $l = 0$, strong reflexivity of the multiset evaluator can not be guaranteed anymore, i.e. it is possible that $E^*_S(s, t) = \{(T, 1)\}$ with $s \neq t$. The parameters of q can be learned on a (relative small) training set by using a search algorithm like steepest descent. After training, the parameterized quantifier contains the knowledge to estimate uncertainty about the co-reference of two strings. A strong benefit regarding performance is that the weight vector does not depend on the strings, but only on the multiset cardinalities. With SoftTFIDF, the weight for each string has to be calculated separately, by look up in a frequency table. On the other hand, SoftTFIDF does not construct an element mapping, which results in performance benefit for SoftTFIDF. Performance tests (Section V) show that the proposed method works faster than SoftTFIDF.

So far, we have presented a study of possibilistic two-level comparison methods for strings due to their benefit for longer strings. However, when doing accuracy tests (Section V) it becomes clear that the benefit of one-level systems is not limited to just detection of errors and abbreviations. More specific, they have the advantage that the order of substrings is respected, which is not the case when comparing multisets of words (this is actually their benefit). Further on, comparing sets is a far more complex operation than comparing strings with an evaluator as defined in Definition 14. These considerations point out that it is useful to combine both $E_{\mathcal{S}}$ and $E^*_{\mathcal{S}}$ into a hybrid evaluator.

Definition 18 (Hybrid string evaluator)

Given an alphabet A and the corresponding universe of strings $\mathcal{S}$. A hybrid string evaluator in $\mathcal{S}$ is defined as:

$$E^+_{\mathcal{S}} : \mathcal{S}^2 \rightarrow \tilde{\wp}(I) : (s,t) \mapsto E^+_{\mathcal{S}}(s,t)$$

where

$$E^+_{\mathcal{S}}(s,t) = f(E_{\mathcal{S},1}(s,t), ..., E_{\mathcal{S},n}(s,t))$$

with f a monotonic ascending function for PTVs and $E_{\mathcal{S},i}$ is an arbitrary evaluator for strings.

In the following we assume a hybrid evaluator based on $E_{\mathcal{S}}(s,t)$ and $E^*_{\mathcal{S}}(s,t)$. To combine both evaluators, we propose generalized disjunction for PTVs defined as:

$$\tilde{\vee} : \tilde{\wp}(I)^2 \rightarrow \tilde{\wp}(I) : \tilde{p}\tilde{\vee}\tilde{q} \mapsto \{(T, u\left(\mu_{\tilde{p}}(T), \mu_{\tilde{q}}(T)\right)), (F, t(\mu_{\tilde{p}}(F), \mu_{\tilde{q}}(F)))\}$$

where t and u are a pair of t-norm/t-conorm. In [19], disjunctive transformations for PTVs that model the impact of $[0,1]$-valued weights are defined, which makes it possible to use a weighted disjunction for f. Doing so has the benefit that the hybrid evaluator can be learned to use a weighted mixture of both evaluators. Such a weighted approach is not further investigated here and is nominated as future work. Instead, a pure disjunctive approach is proposed to study the class of hybrid techniques.

V. Results

A. Test procedure

After introducing evaluators to estimate the uncertainty about the co-reference of strings, accuracy and performance tests are executed in which we compare our techniques with baselines from literature. The performance test measures the execution time of the different methods 20 times and calculates the average execution time. It was found that 20 times was sufficient to provide accurate data. The variance in the measurements is equal for the different methods and an independent samples t-test was used to check whether there is a significant difference between the two fastest methods. The accuracy test procedure assumes two lists L_1 and L_2 containing $< key, value >$-tuples, where the values are the strings to be compared and the keys serve as ground truth. More specific, two tuples with the same key contain co-referent values (strings). Then, for each tuple of L_1, the value is compared with every tuple value in L_2 by using a similarity measure or an evaluator. Hence, for a comparison technique CT, we get a set of 3-tuples $< t_1$,t_2,$CT(v_1, v_2) >$ where t_1 and t_2 are tuples from L_1 and L_2 and v_i is the value of t_i. From this set, an ordered list K can be derived, sorted by $CT(v_1, v_2)$ in descending order. In case $CT(v_1, v_2) = CT(v'_1, v'_2)$, tuples with non-identical keys are ordered first. Next, let dup be the number of duplicates. More specific, dup equals the number of pairs $(t_1, t_2) \in L_1 \times L_2$ for which the keys are identical. We then take the top element of K. If the keys are identical, both the number of correct pairs and the number of found pairs is incremented with one. If the keys are different, only the number of found pairs is incremented. This process is repeated for each element of K, from top to bottom. We then consider the following measures:

$$precision = \frac{correct}{found}$$

$$recall = \frac{correct}{dup}$$

$$F - value = \frac{2 \cdot precision \cdot recall}{precision + recall}$$

The results presented in the following, report the precision at standard recall ratios $\frac{1}{10}, \frac{2}{10}, ..., 1$. Also, the maximum F-value over all recall ratios is reported. The datasets on which the methods are tested are summarized in Table I. The Java library SecondString was used [8] as implementation of the mentioned existing methods. This library is also the one used in [10].

B. Performance test results

To support the statement that E_S is a low complexity comparison method, a comparative test with other character based methods is performed on all datasets, which have varying average string length. We compared the E_S for string evaluation with the following character based

TABLE I
DATASETS

Dataset	**#strings L_1**	**#strings L_2**	**#duplicates**
people	45	45	45
bird1	317	20	19
bird2	914	68	66
bird3	15	23	15
bird4	564	155	155
game	109	798	45
park	396	258	250
restaurant (name)	533	331	112
animal	4719	990	229
census	449	392	327
univ	116	116	676
streets5	1458	585	350
cora	1295	1295	35663
biomed	4000	4000	4992

similarity metrics from literature: Jaro, Jaro-Winkler, Levenstein and Monge-Elkan (i.e. an affine gap method). The reason why these metrics are chosen are because Jaro and Jaro-Winkler are known as the fastest character based methods according to the study in [10]. Levenstein is chosen as a reference, due to it's importance in literature and Monge-Elkan is chosen because our method is an approximation of affine gap methods on a semantical level. Therefor it is interesting to investigate how much faster the new presented method is in comparison with a well known affine gap method. Table II shows the results of our performance test for the character based methods, executed on a Pentium 4 machine with 2.59 GHz CPU and 512 Mb RAM. The fastest method is marked in bold font if it is significantly faster that the second best method. The most important result derived from Table II is the huge difference in speed between the affine gap method and E_S. The new method is clearly a magnitude faster and beats even the Jaro method, which is known in literature as a baseline to combine high accuracy and high performance. In order to illustrate how the complexity of E_S behaves in function of the average string length, we calculate:

$$\frac{execution\ time}{|B| \cdot |C|}$$

TABLE II
RESULTS OF PERFORMANCE TEST ONE-LEVEL METHODS (MS)

Dataset	E_S	**Jaro**	**Jaro-Winkler**	**Levenstein**	**Affine Gap**
people	11	21	21	186	802
bird1	46	87	87	1054	4565
bird2	**256**	492	496	4100	8132
bird3	3	4	5	47	193
bird4	**846**	2117	2151	39275	167500
game	**464**	687	700	9462	18250
parks	**490**	739	752	8987	16984
restaurant (name)	**604**	797	844	8328	16726
animal	**18547**	29750	30182	349062	1449891
census	**1887**	6203	6223	100859	372890
univ	**126**	425	429	6834	29463
streets5	**3131**	4623	4675	45621	90336
cora	**71140**	493171	490985	14155438	67516125
biomed	**69797**	101703	101843	1355625	5546469

for each dataset, which is the average execution time for comparing two strings. The left panel of Figure 4 shows these numbers for each dataset, plotted against the average string length of that dataset. For the sake of simplicity, only Jaro and E_S are plotted. There is a clear linear trend in execution time of E_S, whereas Jaro follows a quadratic trend in function of average string length. Hence, although E_S has a theoretical quadratic complexity in the worst case, it shows linear complexity in practical situations, which is due to the approach of a moving window. Having that the evaluator E_S is indeed a low complexity evaluator, an interesting topic is the complexity of the two-level system E_S^*. In [10] it was noted that the complexity of two-level systems is strongly related to the complexity of the underlying character based method. However, the system we introduced is based on multiset comparison instead of vector comparison and uses a completely different weighting mechanism. Table III shows the results of the performance test for E_S^*, without correction of under tokenization (Section IV) as this is the basic methods, compared with SoftTFIDF. Again, the fastest method that is significantly faster the second best method, is depicted in bold font. In all cases, E_S^* is the fastest method and in most of the cases this difference is significant. However, plotting the execution time of one comparison against the average string length illustrates the quadratic complexity of both two-level method (right panel of Figure 4). Hence, although a low complexity evaluator to compare multiset elements and an

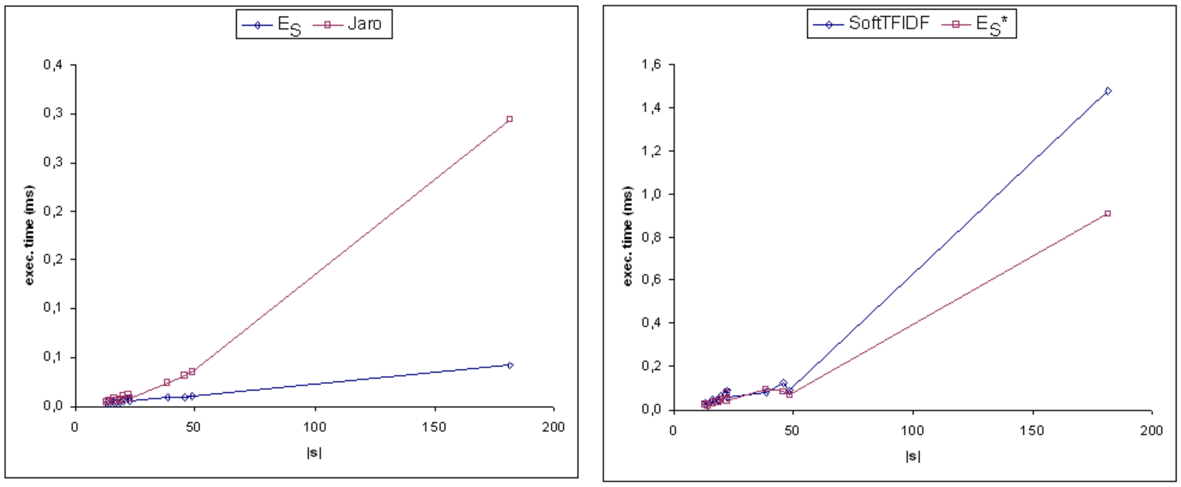

Fig. 4. Complexity in function of average string length for one-level methods (left panel) and two-level methods (right panel)

more efficient weighting scheme have a positive effect on the execution time of the two-level system, the multiset comparison itself remains complex. Therefor, an optimisation of the existing algorithm to create the element mapping ι is an interesting topic for further research. We did not perform accuracy tests for the hybrid method for two reasons. Firstly, the complexity of the hybrid method is clearly the sum of the complexities of the evaluators it uses. The computational effort to calculate the disjunction can hereby be neglected in comparison with the complexity of the evaluators. Secondly, Definition 18 considers that, in general, the number of evaluators used can vary. Therefor, the complexity of a hybrid evaluator is also a matter of how many evaluators are considered.

C. Accuracy test results

In what follows, we study the accuracy of the proposed two-level string evaluator E_S^* and the hybrid evaluator E_S^+. Note that E_S by itself is not the subject of any accuracy test here, because the main topic of interest is creating fast and accurate two-level systems. Of course this requires a fast and accurate character based method. However, we are only interested here in the accuracy of the more advanced systems (E_S^* and E_S^+). As SoftTFIDF is identified as the most accurate method in [7], it is chosen as a baseline method. The low level similarity measure is Jaro-Winkler and the selection threshold is 0.9, which is a commonly used value for this parameter (see [10]). We tested two versions of our two-level technique: (i) OWC with

TABLE III
RESULTS OF PERFORMANCE TEST TWO-LEVEL METHODS (MS)

Dataset	**SoftTFIDF**	$E^*_\mathcal{S}$
people	128	92
bird1	490	**318**
bird2	2325	**1825**
bird3	31	26
bird4	13609	**8354**
game	5257	**3850**
park	4723	**4006**
restaurant (name)	6292	**4839**
animal	215046	**154172**
census	15410	**11895**
univ	1734	**1164**
streets5	17528	**16165**
cora	2480344	**1527172**
biomed	774094	**568328**

$q_{1,0,0.1}$ and (ii) OWC with $q_{\alpha,\beta,l}$ to be trained. Also the hybrid evaluator $E_{\mathcal{S}^+}$ is tested, using a disjunction based on $\min/\max$. The low level evaluator $E_\mathcal{S}$ used by $E_{\mathcal{S}^*}$ has parameter vector $(1,2,3)$, while evaluator $E_\mathcal{S}$ used by $E_{\mathcal{S}^+}$ uses parameter vector $(1,1,1)$ (see Section III). The $E^*_\mathcal{S}$ used by $E^+_\mathcal{S}$ uses a naive quantifier, i.e. $q_{1,0,0.1}$. The conjunction operator used during testing is based on the Lucasiewics t-norm and t-conorm which is given by [18]:

$$t_{luca}(x,y) = \max(x+y-1,0)$$
$$u_{luca}(x,y) = \min(x+y,1)$$

In the case where the parameters are trained, cross validation is performed by randomly selecting 30% of the samples as training set. The remaining 70% of the samples is used as test set. The cross validation is then performed 30 times. In each iteration the maximum F-value is selected. The mean of the 30 maximum F-values is compared with the maximum F-values of the non-trained techniques with a one samples t-test. Besides the maximum F-value we also calculated the average precisions at each recall level in order to construct a precision-recall curve for the trained method. Table IV shows the maximum F-values of the tested methods. Maximum F-values of non-trained methods that significantly differ from the mean of maximum F-values of

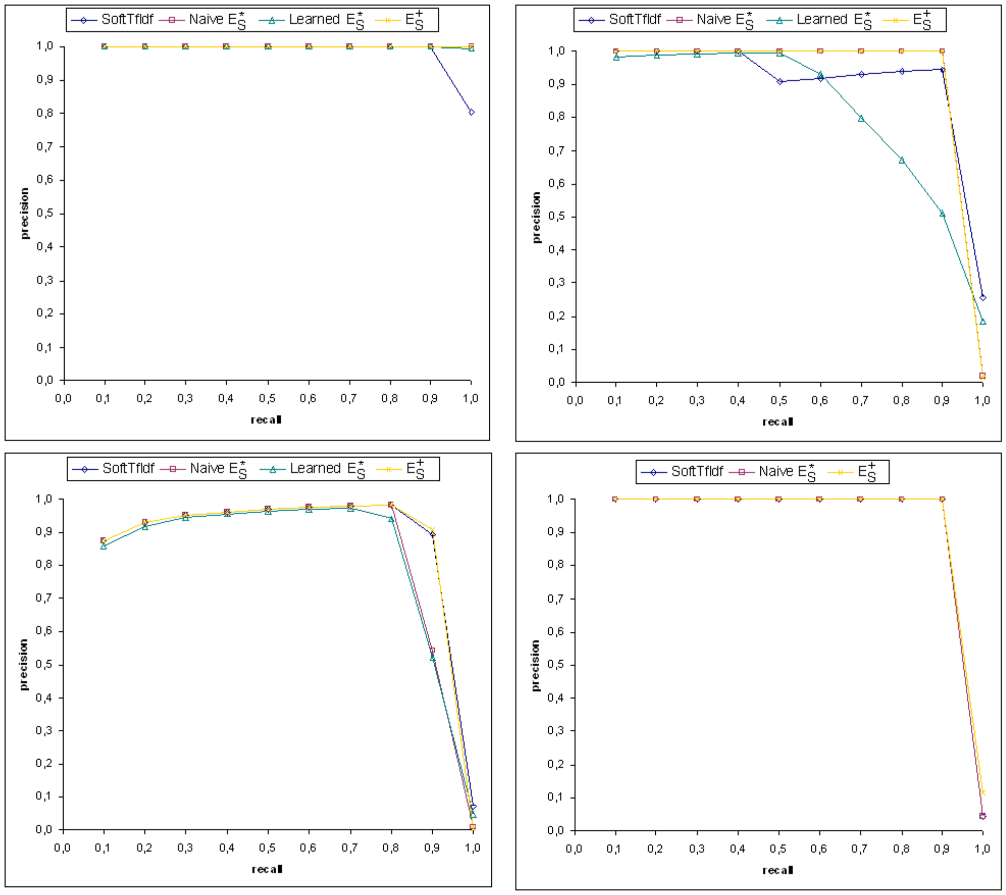

Fig. 5. Precision-recall curve for people (upper left), bird1 (upper right), bird2 (bottom left) and bird3 (bottom right)

the trained method ($p < 0.05$) are depicted in bold font. Figures 5 to 8 show the precision-recall curves for the performed tests. We start by evaluating the naive approach. Table IV shows that naive E^*_S is competitive to SoftTFIDF. For two datasets, the maximum F-value is the same, for six datasets the novel method has a higher maximum F-value and for four datasets, SoftTFIDF has a higher maximum F-value. There is one dataset (people) where our method is a perfect co-reference classifier. The fact that the naive quantifier reflects the required quantity of co-referent substrings well is supported by looking at the learned method. In only two datasets (bird2 and univ), the learned method finds a significant better quantifier than the naive one. Note that for dataset bird2, there is no significant difference between the F-values. For dataset univ, SoftTFIDF is still significantly better, but the learned quantifier is clearly a major improvement in this case (see Figure 7). In two datasets (bird1 and parks), the learned method finds a quantifier

TABLE IV
MAXIMUM F-VALUES

Dataset	**SoftTFIDF**	**Naive** E_S^*	**Learned** E_S^*	E_S^+
people	**0,9474**	1.0000	0.9975	1.0000
bird1	0.9217	**0.9474**	0.8833	**0.9474**
bird2	0.8969	**0.8815**	0.8987	0.9038
bird3	0.9474	0.9474	-	0.9474
bird4	0.9316	0.9442	0.9356	0.9442
game	**0.8276**	0.7535	0.7651	0.7535
parks	**0.9356**	**0.8949**	0.8847	**0.9222**
restaurant	**0.8276**	0.8515	0.8591	**0.8235**
animal	*0.7320*	0.8939	0.8946	**0.8787**
census	**0.6857**	0.8513	0.8520	**0.7350**
univ	**0.8710**	**0.7932**	0.8593	**0.8738**
streets5	0.9474	0.9474	0.9491	**0.9735**
cora	**0.8490**	**0.3318**	0.7870	**0.6804**
biomed	**0.9420**	**0.9425**	0.9221	**0.9381**

that is significantly worser than the naive quantifier, due to overfitting. In other cases, there is no significant difference between the learned method and the naive method. Note that for dataset 'bird3', the learned method could not be applied because the number of samples is too small to support training. A better way to improve naive E_S^* is by extending it to the hybrid method E_S^+. For two datasets (bird2 and univ) where E_S^* performed worser than SoftTFIDF, the hybrid technique performs better than both SoftTFIDF and E_S^*. Also for datasets streets5, where SoftTFIDF and E_S^* have equal maximum F-value, E_S^+ does better. For dataset parks, the accuracy is close to that of SoftTFIDF. A downside is that for three other datasets (restaurant, animal and census), E_S^+ performs worse than E_S^*, however still better than SoftTFIDF. This indicates that extending the used regular disjunction to weighted disjunction is necessary in order to select the correct evaluator depending on the situation. In all datasets, the tokenization step uses non alphanumerical characters as delimiters. For dataset 'biomed', our method uses correction of under tokenization by concatenating tokens where needed. The threshold used in this step was $\{(T, 1)\}$ (see Section IV). It is noted that SoftTFIDF is competitive with our method without using such a correction step, which is caused by the fact that SoftTFIDF uses addition to combine evidence, while the proposed method uses conjunction. An interesting dataset to discuss is the cora dataset, which is the only dataset where SoftTFIDF has a clear benefit over the novel two-

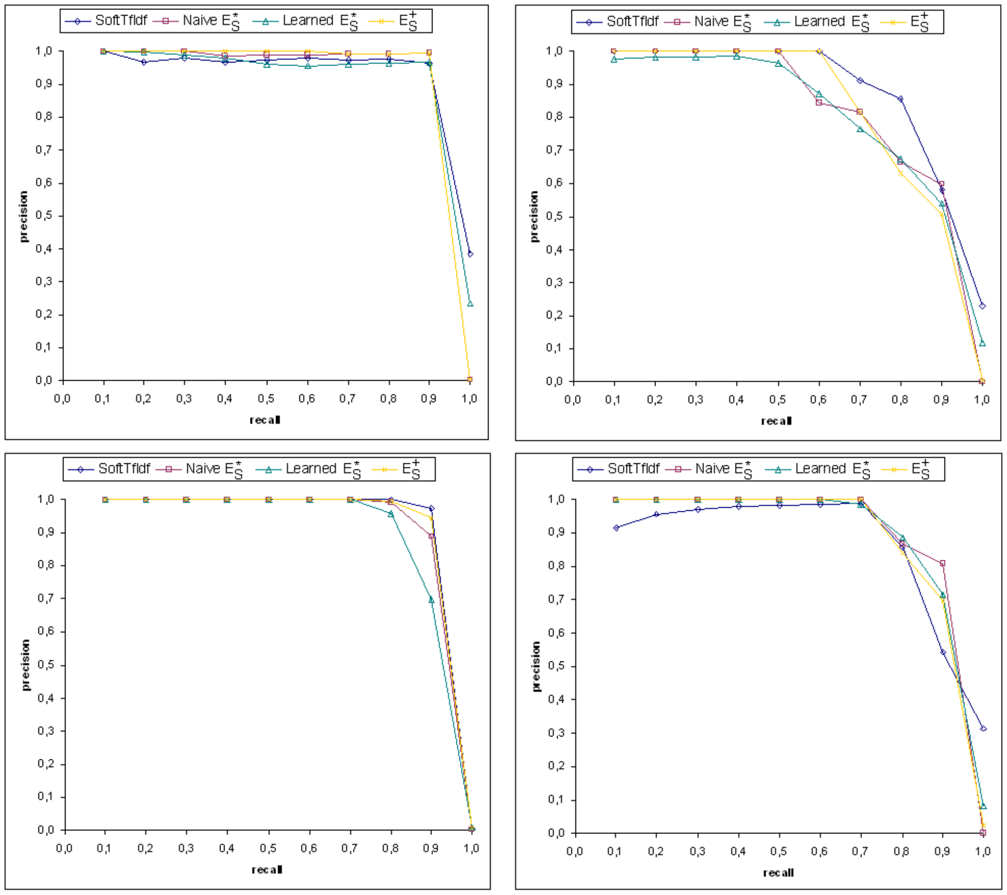

Fig. 6. Precision-recall curve for bird4 (upper left), game (upper right), parks (bottom left) and restaurant (bottom right)

level method. This is due to the nature of the data, which are references. High frequent words (such as “in”, “and”, author names,...) distort the benefit of the quantifier. In this case, it might be fruitful to think of an approach where the best of two worlds is combined. Although such an approach is outside the scope of this paper, we will illustrate it briefly. The key point is to drop the reflexivity of the low level evaluator E_S. More specific, two strings that are equal are not necessarily co-referent. This is due to the uncertainty that is caused by the fact that the strings are highly frequent throughout the dataset. To model this behavior, assume for a string s that f_s is the frequency of this string in the dataset and let $|D|$ denote the size of the dataset. Define:

$$u_s = \frac{f_s - 1}{|D| - 1}$$

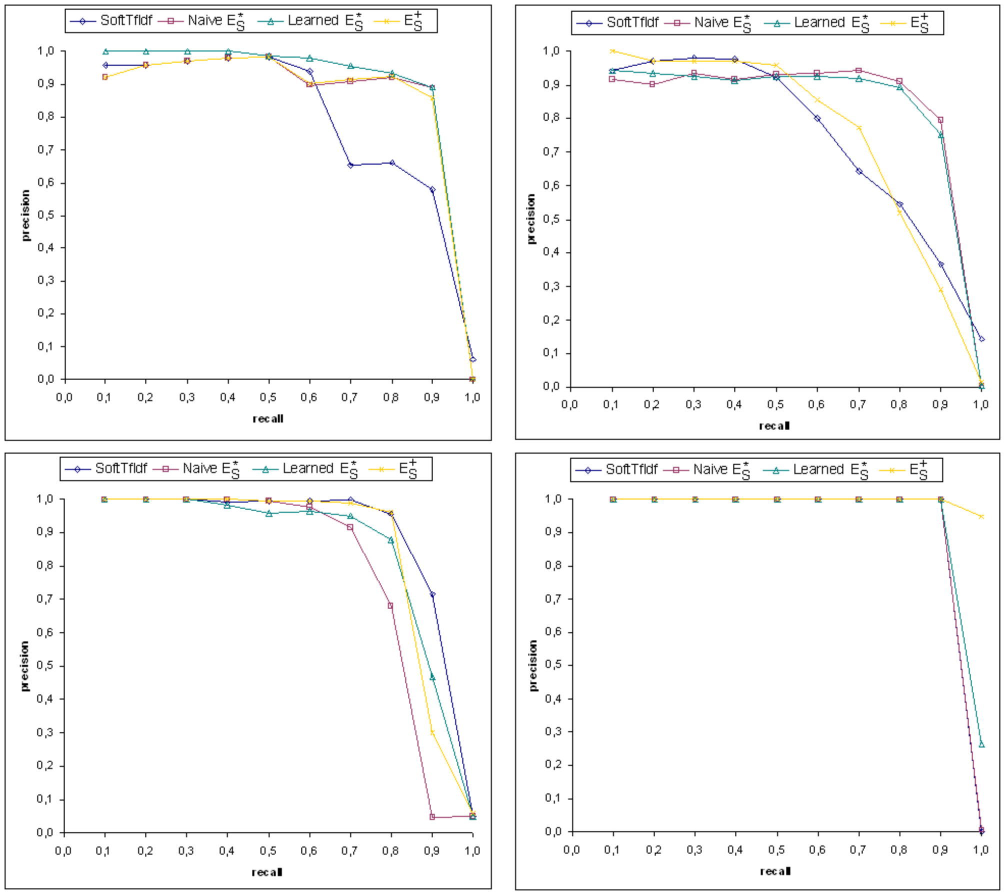

Fig. 7. Precision-recall curve for animal (upper left), census (upper right), univ (bottom left) and streets5 (bottom right)

This leads to a modification of $E_{\mathcal{S}}$ as follows:

$$E_{\mathcal{S}}^{m}(s,t) = \left\{ \left(T, \mu_{E_{\mathcal{S}}(s,t)}(T)\right), \left(F, \max\left(\mu_{E_{\mathcal{S}}(s,t)}(T), u_s, u_t\right)\right) \right\}$$

Using such an evaluator takes into account additional uncertainty based on the frequency of words. In an extreme case, if a string s occurs in each sample of the dataset (e.g.: s="the"), then $u_s = 1$ and $E_{\mathcal{S}}^{m}(s,s) = \{(T,1),(F,1)\}$ whereas $E_{\mathcal{S}}(s,s) = \{(T,1)\}$. A consequence of this approach would be that the complexity increases significantly because (i) the intersection of the multisets can no longer be treated separately and (ii) u_s must be calculated for each token. Further on, the role of quantifier parameters becomes much more important, due to the increased uncertainty on the lower level. A further evaluation of this method is therefor outside the scope of this paper.

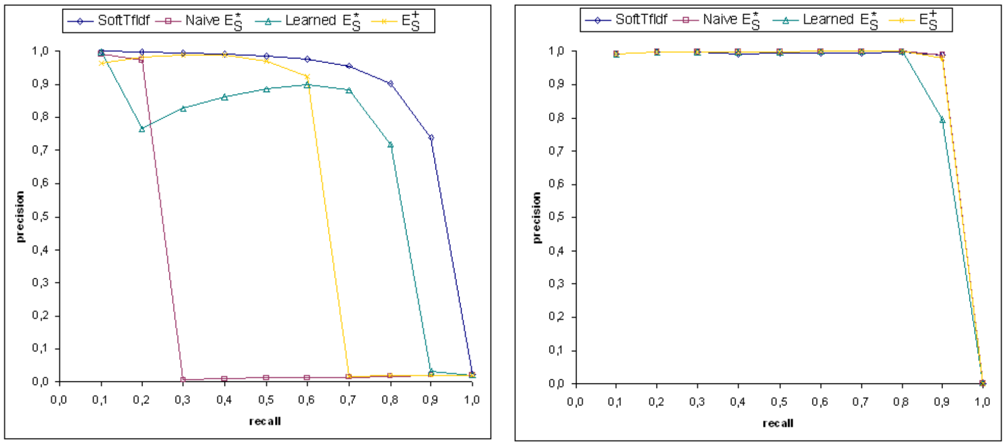

Fig. 8. Precision-recall curve for cora (left panel) and biomed (right panel)

VI. Future work

In the previous Section, possibilistic approaches on string comparison are introduced. Section V showed that these methods are competitive (with respect to accuracy and performance) with a strong baseline method from literature. However, it showed also that there are some points in which the proposed ideas can be improved. A first illustration was already given while discussing the results on the cora dataset, where a modification of Definition 14 (E_S) was proposed to deal with situations like the cora dataset. A second thing learned from discussing the results is the variating performance of the hybrid evaluator, due to the pure disjunctive combination function. It is certainly worth while to use a weighted disjunctive function that selects the appropriate evaluator depending on the application. It follows that a learning algorithm would be necessary to provide these weights. Third, the proposed quantifier is only one of the possible solutions to provide a weight vector. Other quantifiers, such as absolute quantifiers defined with respect to a given number should be investigated. In the same direction, it was shown in [19] that OWC has interesting properties with respect to optimisation. Using these properties in the current context can provide a reduction in complexity and hence also in execution time. A last point that requires future research is the investigation of the cost model used by E_S. The model proposed in this paper is a very simple one and shows good results. However, it is very likely that other models exists, that perform better than the one introduced here.

VII. Conclusion

A possibilistic view on string comparison is given by proposing operators that provide the possibility of co-reference of two strings. These possibilities are presented by means of possibilistic truth values. First, a fast character based method is proposed based on the properties of weak string intersection. Next, this method is applied in a possibilistic two-level approach on string comparison. The use of ordered weighted conjunction in combination with a parameterized fuzzy quantifier is illustrated and it is shown that such an approach is competitive with a strong baseline method from literature. In addition, hybrid evaluators are proposed that combine character based and two-level evaluators to infer a final result by use of a monotonic combination function. It is shown that a pure disjunction results in a variating accuracy, depending on the application. To solve this problem weighted disjunction is proposed.

References

[1] Vladimir Levenshtein, "Binary codes capable of correcting deletions, insertions and reversals," *Physics Doklady*, vol. 10, no. 8, pp. 707–710, 1966.

[2] Frederik Damerau, "A technique for computer detection and correction of spelling errors," in *Communications of the ACM*, 1964.

[3] Saul Needleman and Christian Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *Journal of Molecular Biology*, vol. 48, no. 3, pp. 443–453, 1970.

[4] Michael Waterman, Temple Smith, and William Beyer, "Some biological sequence metrics," *Advances in Mathematics*, vol. 20, no. 4, pp. 367–387, 1976.

[5] Alvaro Monge and Charles Elkan, "The field matching problem: Algorithms and applications," in *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, 1996, pp. 267–270.

[6] Matthew Jaro, "Unimatch: A record linkage system: Users manual," US Bureau of the Census, Tech. Rep., 1976.

[7] William Cohen, Pradeep Ravikumar, and Stephen Fienberg, "A comparison of string distance metrics for name-matching tasks," in *Proceedings of the IJCAI Workshop on Information Integration on the Web*, 2003, pp. 73–78.

[8] ——, "Secondstring: an open source java toolkit of approximate string-matching techniques."

[9] William Cohen, "Integration of heterogeneous databases without common domains using queries based on textual similarity," in *Proceedings of the ACM Sigmod International Conference of Management of Data*, 1998, pp. 201–212.

[10] Mikhail Bilenko, Ray Mooney, William Cohen, Pradeep Ravikumar, and Stephen Fienberg, "Adaptive name matching in information integration," *IEEE Transactions on Intelligent Systems*, vol. 18, no. 5, pp. 16–23, 2003.

[11] Luis Gravano, Panagiotis Ipeirotis, Nick Koudas, and Divesh Srivastava, "Text joins in an rdbms for web data integration," in *Proceedings of the International World Wide Web Conference*, 2003, pp. 90–101.

[12] Antoon Bronselaer, Axel Hallez, and Guy De Tré, "Evaluation in the possibilistic framework for object matching," in *Proceedings of the IPMU 08 Congres*, Malaga, 2008.

[13] Lotfi Zadeh, "Fuzzy sets as a basis for a theory of possibility," *Fuzzy Sets and Systems*, vol. 1, pp. 3–28, 1978.

[14] Henri Prade, “Possibility sets, fuzzy sets and their relation to lukasiewicz logic,” in *Proceedings of the International Symposium on Multiple-Valued Logic*, 1982, pp. 223–227.

[15] Didier Dubois and Henri Prade, *Possibility Theory*. Plenum Press, 1988.

[16] Gert De Cooman, “Towards a possibilistic logic. in: Fuzzy set theory and advanced mathematical applications, edited by d. ruan, kluwer academic, pp. 89–133, boston.”

[17] Lotfi Zadeh, “Fuzzy sets,” *Information and Control*, vol. 8, pp. 338–353, 1965.

[18] Didier Dubois and Henri Prade, *Fundamentals of Fuzzy Sets*. Kluwer Academic, 2000.

[19] Antoon Bronselaer and Guy De Tré, “Impact of [0,1]-valued weights and weighted aggregation operators for possibilistic truth values,” in *Proceedings of the NAFIPS 08 Congres*, New York, 2008.