Effective Unsupervised Matching of Product Titles with k-Combinations and Permutations

Leonidas Akritidis
Data Structuring & Engineering Lab
Dept. of Electrical and Computer Engineering
University of Thessaly
Volos, Greece
leoakr@e-ce.uth.gr

Panayiotis Bozanis
Data Structuring & Engineering Lab
Dept. of Electrical and Computer Engineering
University of Thessaly
Volos, Greece
pbozanis@e-ce.uth.gr

***Abstract*—The problem of matching product titles is of particular interest for both users and marketers. The former, frequently search the Web with the aim of comparing prices and characteristics, or obtaining and aggregating information provided by other users. The latter, often require wide knowledge of competitive policies, prices and features to organize a promotional campaign about a group of products. To address this interesting problem, recent studies have attempted to enrich the product titles by exploiting Web search engines. More specifically, these methods suggest that for each product title a query should be submitted. After the results have been collected, the most important words which appear in the results are identified and appended in the titles. In the sequel, each word is assigned an importance score and finally, a similarity measure is applied to identify if two or more titles refer to the same product. Nonetheless, these methods have multiple problems including scalability, slow retrieval of the required additional search results, and lack of flexibility. In this paper, we present a different approach which addresses all these issues and is based on the morphological analysis of the titles of the products. In particular, our method operates in two phases. In the first phase, we compute the combinations of the words of the titles and we record several statistics such as word proximity and frequency values. In the second phase, we use this information to assign a score to each combination. The highest scoring combination is then declared as label of the cluster which contains each product. The experimental evaluation of the algorithm, in a real world dataset, demonstrated that compared to three popular string similarity metrics, our approach achieves up to 36% better matching performance and at least 13 times faster execution.**

***Index Terms*—products matching, entity matching, unsupervised learning, data mining, algorithms**

I. Introduction

In the past few years, the online product comparison platforms have gained the attention of the consumers. These platforms receive data—usually in the form of feeds—from multiple sources, including electronic stores, reviews sites and aggregation applications. In the sequel, they process this diverse data and they unify the relevant information which refers to the same products. Finally, they present this information to their users, allowing them to compare a wide variety of parameters such as prices, availabilities, delivery methods, delays, costs, guarantees, overcharges, and numerous others. They also facilitate the aggregation of user opinions, reviews, and purchase decisions.

Since the data is provided by multiple sources, it presents a high degree of diversity. In most cases, the same products appear under different titles and descriptions. Furthermore, some vendors offer richer data than others by including more information in their product records such as brands, categories, technical specifications and others. Consequently, to provide a unified view of the product, there must be a procedure which matches these different product records. Apparently, the problem of matching product records from different data feeds is vital for these platforms, their users, and online shopping in general.

Due to the importance of the problem, there is currently a vast amount of research in the relevant literature. This research can be divided into two categories: In the first category, we mainly encounter works which attempt to solve the problem by examining the titles of the products only. On the other hand, the second category includes methods which take into consideration additional overlapping attributes, such as brands, manufacturers, categories, etc. The algorithm we introduce in this article belongs to the first category. Since a product comparison platform may receive data from thousands of non controlled sources, many of the attributes that are present in one feed may be absent in another. In such occasions, it is inevitable that the methods of the latter category will not achieve a satisfactory performance.

Earlier works have focused on the utilization of the traditional string similarity metrics such as the edit distance metric, the Jaccard similarity, the cosine similarity, and others. However, as [1] demonstrates, none of these metrics performs well for this particular problem. Indeed, consider the titles *Apple iPhone 7* and *Apple iPhone 7 plus*. Although the similarity of these two titles is high, they refer to the different products. On the other hand, the similarity of the titles *Apple iPhone 7 black 32gb* and *iPhone 7* is lower; however, they both describe the same device.

To overcome the poor performance of the string similarity metrics, the authors of [1] employed Web search engines with the aim of enriching the involved product titles. For each product title t, their method submits a query t to a search engine and collects the results. In the sequel, it identifies the most important words appearing in the results, and it utilizes them to obtain an enriched form of the title. Finally, it assigns an importance score to each word based on their ability to retrieve other tokens in search results (the algorithm submits another set of queries). However, the submission of a query in a Web search engine and the subsequent processing of the returned results, is an expensive procedure. Furthermore, the APIs provided by the search engines for this task do not allow unlimited usage of their resources, and there is an upper bound to the number of the queries which can be submitted on a daily basis. Even though the authors propose an optimization which reduces the number of queries to be submitted, in case of large-scale datasets with millions of products, the method is rendered infeasible. In [2], the authors extended this algorithm by modeling the titles as graphs. In the sequel, a clustering algorithm examines whether these two graphs form a cohesive community, or separately clustered communities. Nevertheless, this method also suffers from the aforementioned problem caused by the usage of Web search engines.

In this article we introduce a two-phase unsupervised algorithm which achieves much better performance than the traditional similarity metrics, and it is also independent of any external sources like Web search engines. During the first phase, a tokenizer parses the titles of the products and computes all the possible combinations of the words of the titles. These combinations, which vary in length, are stored within a lexicon, accompanied by their frequency in the corpus, and a proximity value which records the distance of the combination from the beginning of the title.

Since the words of a combination may appear in a different order within a product title, we also compute the permutations of these combinations to capture their similarity. In this phase, we additionally build a second data structure, the forward index, which maintains for each product, a list of pointers to all the corresponding word combinations appearing in its title. In the second phase, the algorithm traverses all the lists in the forward index and computes a score for each combination and for each product. This score is calculated by a function which embodies the frequency of the combination, its length, and its average distance from the beginning of the titles which contain it. At the end of this process, the highest scoring combination is declared as the representative cluster of the product.

Notice that, in contrast to the competitive methods, our proposed algorithm does not perform direct pairwise comparisons of the product titles. Instead, it incrementally constructs two data structures and during this phase, it updates several local and global statistics, such as the frequency, and the distance of a word combination from the beginning of the title. This strategy, combined with the avoidance of the Web search engines (or any other external data source), reduces the complexity and boosts the efficiency of our approach. Moreover, in contrast to the traditional similarity metrics, our algorithm does not require the introduction of an accompanying blocking method to reduce the number of the comparisons.

The rest of the paper is organized as follows: In Section II we refer to some of the most significant related work in the current literature. In Section III we provide a brief overview of the basic parameters of the problem, whereas in Section IV we describe in details the proposed algorithm for matching product titles. The experimental demonstration of the usefulness of the algorithm is presented in Section V. Finally, in Section VI we conclude the paper, and in Section VII we briefly describe the future work which shall further improve the method and its applications.

II. Related Work

Due to its importance, the wide problem of string matching has attracted many researchers and it has been extensively studied. A considerable number of works employ the traditional string similarity metrics such as the edit distance metric, cosine similarity, Jaccard similarity, n-grams, atomic strings, etc [3]–[8]. However, in the context of matching product titles, they have been proved rather ineffective, as indicated by [1]. Moreover, the studies conducted in [4], [9] and [10] present systematic and complete surveys of the relevant state-of-the-art research in the area.

Another approach for solving this problem is to introduce methods which take into consideration the additional attributes of an entity. In the case of products, these additional attributes include brands, categories, prices, etc. The examined problem usually appears under multiple different names in the literature, like *entity matching* [11]–[13], *entity disambiguation* [14]–[16], *duplicate detection* [17], [18], or *near duplicate detection* [19], [20].

Furthermore, a multitudinous family of methods is based on machine learning learning algorithms. In [21] the authors introduced an unsupervised method which extended the Latent Dirichlet Allocation model and proposed a probabilistic model for collective entity resolution. Regarding the supervised learning approaches, FEBRL provides an implementation based on SVMs with the aim of learning suitable matcher combinations [22]. On the other hand, MARLIN offers a set of several learning methods such as SVMs and decision trees, combined with two string similarity measures (edit distance and cosine similarity) [17]. A comparative study of these methods is presented in [3], where the authors conclude that the traditional supervised learning approaches result in low matching quality for products. Another interesting outcome of this study indicates that the methods which take into consideration multiple attributes generally outperform those which are based solely on the processing of the titles. Nevertheless, as mentioned earlier, the product attributes are not always provided by the data vendors, or even if they are present, they may be inaccurate. Consequently, these methods do not perform well in such cases.

Finally, the aforementioned methods described in [1] and [2] employ the Web search engines with the aim of enriching the product titles with important missing words. Then, the former assigns importance scores to each word in the title, whereas the latter models the titles as graphs and applies a clustering algorithm to examine their correlation. The main weakness of these methods is that they both suffer from scalability issues, since they are obliged to tolerate the expensive and slow retrieval of the required search results.

III. Overview

Let us consider a set of input source feeds

$$S = \{s_1, s_2, \ldots\},$$

where each source feed s consists of a set of product records

$$P_s = \{p_{s,1}, p_{s,2}, \ldots\}.$$

Each product record contains an arbitrary number of attributes including its title, price, brand, category, and others. Notice that a feed is formatted independently of the others; consequently, a feed s_i may include information about the brand, or the category of a product, whereas s_j may not. However, all input feeds must provide a descriptive title t for each included product.

In each feed, the vendor may use a different title to describe the same product. For instance, one may use *Apple iPhone 7*, whereas another may merely use *iPhone 7*, both describing the same mobile device. The problem that we examine in this article is to devise a method which will decide correctly if these two (or more) different titles describe the same product. We also require to group these products under the same cluster with a descriptive label, with the aim of enabling robust comparison and aggregation.

IV. Unsupervised Titles Matching

In this Section we analyze the most important parts of our proposed algorithm. Initially, we present some preliminary elements and, in the sequel, we describe the two phases involved in the method.

Given a string t, we consider a set T which consists of all the words of t. Now a k-combination is any subset of T of length k, without repetition and without care for word ordering.

For instance, if

$$T = \{t_1, t_2, t_3\}$$

then there are three possible 2-combinations, namely

$$\{t_1, t_2\}, \{t_1, t_3\}, \{t_2, t_3\}$$

and only one 3-combination

$$\{t_1, t_2, t_3\}.$$

It is clear that if t consists of l_t words (i.e., its length is l_t), then the number of all possible k-combinations is:

$$\binom{l_t}{k} = \frac{l_t!}{k!(l_t - k)!} \qquad (1)$$

Notice that k-combinations are different than the standard n-grams which are commonly used in the literature: the n-grams are computed by sliding a ‘window‘ of length n over the examined string, therefore, they contain only successive words. On the other hand, k-combinations contain both adjacent and non-adjacent words.

Although k-combinations are more numerous than n-grams, we choose to construct them instead of dealing with n-grams, since, frequently, the important words appear scattered across the titles and also, in non adjacent positions.

For example, there is no common 2-gram or 3-gram for the titles

Intel CoreI7 7700K 3.6GHz

and

CoreI7 3.6GHz 7700K,

whereas there are two common 2-combinations, namely

CoreI7 7700K and *CoreI7 3.6GHz.*

Consequently, we consider that k-combinations are preferable over n-grams for this particular problem.

In the analysis which follows, we also employ the notion of *permutations*. Given a k-combination, the set of permutations M consists of all the $k!$ possible orderings of the words of that combination.

In the previous example, the computation of permutations will additionally capture the similarity between the 2-combinations

7700K 3.6GHz and *3.6GHz 7700K*

and the 3-combinations

CoreI7 7700K 3.6GHz and *CoreI7 3.6GHz 7700K.*

Notice that we desire to keep the number of combinations to a minimum, because each new combination increases the complexity of the solution. Since most products are sufficiently described by using only a few K words (provided that we remove the unnecessary terms such as offer descriptions, and/or technical specifications), we limit the computations to the first $2, 3, ..., K$-combinations of the words of the involved titles.

Therefore, for a title which consists of l_t words the total number of combinations to be computed is:

$$|C| = \sum_{k=2}^{k=K, k \leq l_t} \binom{l_t}{k} = \sum_{k=2}^{k=K, k \leq l_t} \frac{l_t!}{k!(l_t - k)!} \qquad (2)$$

Regarding the permutations, the total number which can be computed is:

$$|M| = \sum_{k=2}^{k=K, k \leq l_t} \frac{l_t!}{(l_t - k)!} \qquad (3)$$

Apparently, this number is considerably larger than the number of the combinations given by eq. 2. However, as we explain in subsection IV-A, the permutations are only required under special circumstances. Consequently, we rarely have to compute them all.

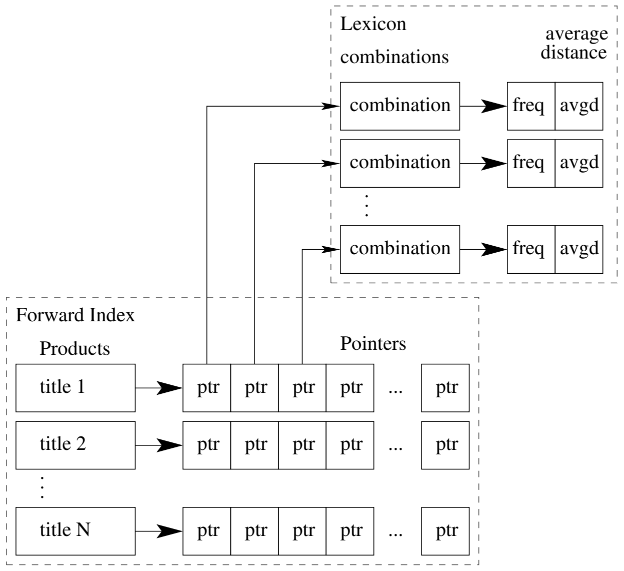

Fig. 1. The interconnection of the forward index with the lexicon.

A. Phase 1: Titles processing and data structures construction

The first phase of our proposed algorithm includes the processing of the titles of the involved products and the construction of two data structures which shall assist us in the identification of the appropriate product clusters.

The procedure starts with the initialization of these two data structures:

- A lexicon L or *combinations dictionary*, which is used to store all the k-combinations extracted from the titles of the products. In addition, for each k-combination c, the lexicon also maintains:
  i) a frequency value which is used to store the number of products which contain this combination; and
  ii) a distance accumulator which accommodates the sum of the distances of c from the beginning of the titles.
- A forward index F, which, for each product p, stores a list of $|C|$ records (recall that $|C|$ is computed by using eq. 2). Each of these records contains one pointer which points to the respective k-combination contained within p, and a score value which will indicate the strength of the correlation of the k-combination with the product. This score is initially set equal to zero, and it shall be computed during the second phase.

In Figure 1 we depict the interconnection of the forward index with the lexicon structure. Notice that the existence of pointers in the forward index saves us the cost of storing the same data twice.

Algorithm 1 describes the procedure which constructs these two data structures. Initially, each product title passes through a linguistic processing function (step 5) which performs case folding—i.e., conversion of capital characters to lowercase—and removal of the punctuation symbols including dots, commas, parentheses, etc. Since hyphens and slashes are usually encountered in model descriptions, they are left intact by the punctuation removal filter.

Algorithm 1: Product titles' processing and data structures construction

```
1  initialize the lexicon $L$;
2  initialize the forward index $F$;
3  for each product $p$ do
4      extract the title $t$;
5      perform linguistic processing of $t$;
6      for each $k \in [2, K]$ do
7          compute all $k$-combinations $C_{(k)}$ of $t$;
8          for each $k$-combination $c \in C_{(k)}$ do
9              Set $d(c,t) \leftarrow$ distance($c$,$t$);
10             $F$.insert($p$,$c$);
11             Set $found \leftarrow L$.search($c$);
12             if $found$ = true then
13                 Set $c$.freq $\leftarrow$ $c$.freq + 1;
14                 Set $c$.dist $\leftarrow$ $c$.dist + $d(c,t)$;
15             else
16                 compute all permutations $M$ of $c$;
17                 for each permutation $m \in M$ do
18                     Set $found \leftarrow L$.search($m$);
19                     if $found$ = true then
20                         Set $c$.freq $\leftarrow$ $c$.freq + 1;
21                         Set $c$.dist $\leftarrow$ $c$.dist + $d(c,t)$;
22                         break;
23                     end
24                 end
25             end
26             if $found$ = false then
27                 $L$.insert($c$);
28                 Set $c$.freq $\leftarrow 1$;
29                 Set $c$.dist $\leftarrow d(c,t)$;
30             end
31         end
32     end
33 end
```

In the sequel, we create all $2, 3, ..., K$-combinations of t and, for each combination c, we compute a distance between the strings c and t (step 9). This distance value will be used later by the scoring function with the aim of assigning higher scores to the combinations which appear early in the title. The reason that justifies the computation of this distance value originates from the intuition that the most important words in a product title appear early, that is, in small positions, and should be treated differently than the words which appear in higher positions. In Subsection IV-B we present the function which we employed to compute this distance, and we provide a detailed discussion about it.

After $d(c,t)$ has been computed, we insert a pointer to c in the appropriate list of p in the forward index F (step 10), and we search for c in the lexicon L (step 11). In case the search is successful (i.e., c already exists in L, steps 12–14), we increase

Algorithm 2: Scores computation and cluster selection

```
1 for each product $p$ in $F$ do
2     retrieve the forward list $f_p$;
3     for each $c \in f_p$ do
4         Set $c$.adist ← $c$.dist / $c$.freq;
5         Set $c$.score ← ComputeScore($c$);
6     end
7     sort $f_p$ in decreasing score order;
8     Set cluster ← $f_p[0]$;
9 end
```

Dataset 1	**Dataset 2**
Apple iPhone 7 black	Apple iPhone 7 black (retail)
Apple iPhone 7 gold	Apple iPhone 7 gold (retail)
Apple iPhone 7 silver	Apple iPhone 7 32gb silver
Apple iPhone 6 black	Apple iPhone 6 black (retail)
Apple iPhone 6 gold	Apple iPhone 6 gold (retail)
Apple iPhone 6 silver	Apple iPhone 6 32gb silver
Apple iPhone 6s black	Apple iPhone 6s black (retail)
Apple iPhone 6s gold	Apple iPhone 6s gold (retail)
Apple iPhone 6s silver	Apple iPhone 6s 32gb silver

TABLE I
HYPOTHETIC DATASETS

its frequency by one, we update the distance accumulator and we proceed to the next combination. Otherwise, c does not exist in L, so we compute all its permutations M and, for each $m \in M$, we perform a search in the lexicon for m (steps 15–25). If this search is successful (steps 19–23), we similarly increase the frequency of $c \in L$, we update the distance accumulator, and we proceed to the next combination. Notice that the permutations are only computed when a combination is not present in L, that is, only at its first occurrence. Consequently, the total number of the computed permutations is considerably smaller than the number indicated by equation 3. Furthermore, no permutation is inserted into the lexicon; what is inserted is the originating k-combination. This strategy limits the size of the lexicon and prevents it from becoming unacceptably large.

In case neither the k-combination c nor any of its permutations exist in the lexicon L, we insert it in L and we set its frequency equal to 1 (steps 24–27). Then, we proceed to the next combination. At the end of this process, all product titles have been examined, we have the L and F data structures fully constructed, and we are ready to apply the algorithm of the next phase.

B. Phase 2: Scores Computation and Cluster Selection

The ultimate goal of this phase is to compute the scores for each combination record c in the forward index F, with respect to the product title t which contains it.

This part of the algorithm is quite straightforward; we iterate through the product records of F and we access their forward lists. Recall that the forward list of a product p stores one pointer for each k-combination c included in p, and a score value $S(c)$ which shall be calculated during this phase. In step 4 we compute the average distance value $\overline{d(c)}$, merely by dividing the aforementioned distance accumulator by the frequency of c. After all score values have been computed, we sort the forward list of each product by decreasing score order and we select the first (that is, the highest scoring) entry to represent the desired product cluster. The steps of this procedure are presented in Algorithm 2.

We shall now discuss in more details the selection of the scoring function of step 5. At first, we study the basic properties that a k-combination should possess to be declared as the representative product cluster, and then we proceed to the quantification of these properties. To make the following notions clearer, we shall use the hypothetic datasets of Table I.

- *Frequency*: Our goal is to identify a cluster with an appropriate label to classify all the same products under it. This means that a k-combination must appear in as many product titles as possible. Otherwise, in case we select a rare k-combination, we shall not be able to cluster any other product under it. This requirement justifies the assignment of frequency values to each k-combination within the lexicon L.
- *Length*: The frequency criterion definitely favors the short k-combinations, since it is more possible to encounter a combination of two words, than a combination of three words. However, the short k-combinations are not as descriptive as the longer ones and, furthermore, there is a risk of creating very generic and inhomogeneous clusters which may erroneously contain different products. Consequently, whenever it is possible, we desire to boost the score of the longer k-combinations.
  This is better perceived if we consider the hypothetic Dataset 1 of the left column of Table I: Here, the most frequent combination is obviously *Apple iPhone*. However, it would be wrong to be declared as the representative cluster of all 9 products. The correct solution is to identify 3 different clusters with labels *Apple iPhone 7*, *Apple iPhone 6*, and *Apple iPhone 6s*. Although their frequency is lower than that of *Apple iPhone* (3 instead of 9), they are more descriptive and they correctly identify the three different products of the dataset.
- *Position*: A broadly accepted idea in information retrieval dictates that the most important words in a document usually appear early, that is, in a small distance from its beginning. The proposed method also adopts this idea for the k-combinations.
  Given a title t, a k-combination c of t, and a word $w \in c$, we consider that $o_w^{(c)}$ is the position (or offset) of w in c and $o_w^{(t)}$ is the position of w in t. Based on these notations, we compute the distance $d(c,t)$ between c and t by employing the well-established Euclidean distance for strings:

$$d^2(c,t) = \sum_{w \in c} \left(o_w^{(c)} - o_w^{(t)} \right)^2 \qquad (4)$$

The motivation which led us to the introduction of this distance-based scoring approach, is the frequent occurrences of words with no informational value.

This is made clear in the hypothetic Dataset 2 of the right column of Table I: In this example, the most common 3-combination is *Apple iPhone retail* which apparently is an inappropriate cluster label. The problem is caused by the word *retail*, a frequent word with no informational value. We have examined a large number of product titles and we identified multiple words with such properties, including *OEM*, *EU*, *offer*, or even product colors (black, silver, etc.). The key element with all these words is that they usually occur in high positions, that is, in large distances from the beginning of the title.

Based on the three aforementioned properties, we now present the function which is used to compute the score of a k-combination c:

$$S(c) = \frac{l_c}{\alpha + \overline{d(c)}} \log N_c \tag{5}$$

where $l_c > 1$ is the length (in words) of c, N_c is the number of the products which contain c, and $\overline{d(c)}$ represents the average distance of c from the beginning of the titles which contain it, that is:

$$\overline{d(c)} = \frac{1}{N_c} \sum_{\forall t \subset c} d(c,t) \tag{6}$$

Moreover, $\alpha > 0$ is a constant quantity with a dual role: i) it prevents the score $S(c)$ from approaching infinity when $\overline{d(c)} = 0$ (i.e, when c appears always in the beginning of all titles); and ii) it regulates the contribution of proximity in the overall score of a k-combination.

Consequently, the scores of eq. 5 indicate that a product should be clustered under a label which is frequent, reasonably long, and, also, its words usually occur near the beginning of the titles of the products.

V. Experiments

Here we demonstrate the robustness of the proposed method in terms of both effectiveness and efficiency. All experiments were performed on a workstation equipped with a single Intel CoreI7 7700@3.6GHz CPU (single-core mode) and 32GB of RAM, running Ubuntu Linux 16.04 LTS.

The dataset we used was acquired by deploying a focused crawler on a subset of a popular product price comparison platform in Greece, skroutz.gr. More specifically, we crawled the section which contains the mobile devices of the platform and we constructed a database of 16208 products provided by 320 electronic stores.

To facilitate prices and features comparison, the platform itself groups (i.e., matches) the same products in clusters. Regarding the mobile devices subset, the total number of clusters is 922; this means that, on average, each cluster contains 17.6 products, or, equivalently, each product appears on average under 17.6 different titles. The average title length is roughly 8.97 words, whereas the longest title includes 23 words. This dataset establishes a particularly challenging scenario due to its size (it is several times larger than other similar datasets), and due to the high diversity of the titles which originates from the large number of the included product vendors.

In the following analysis of the experimental results, we will refer to the proposed algorithm by using the abbreviation *UMaP (Unsupervised Matching of Products)*. UMaP is compared against three traditional IDF-based string similarity metrics, in a spirit similar to other works such as [1].

The first metric we used for our comparisons is the well-established cosine similarity. According to it, given two strings t_1 and t_2, with respective lengths (in words) $|t_1|$ and $|t_2|$, their similarity is determined by the following equation:

$$cos(t_1, t_2) = \frac{|t_1 \cap t_2|}{\sqrt{|t_1|}\sqrt{|t_2|}} \tag{7}$$

The second metric which we included in our evaluation is Jaccard similarity (*JSim*) [23], which quantifies the similarity between two strings as follows:

$$J(t_1, t_2) = \frac{|t_1 \cap t_2|}{|t_1 \cup t_2|} \tag{8}$$

Finally, we also employed the Jaro-Winkler distance (*JRD*) [24], [25], an edit distance metric which essentially represents the minimum number of single-character transpositions required to change one word into the other. If x is the number of the matching characters between t_1 and t_2, and y is half the number of transpositions, then the Jaro-Winkler distance is calculated according to the following branch equation:

$$JR(t_1, t_2) = \begin{cases} 0 & \text{if } x = 0 \\ \frac{1}{3}\left(\frac{x}{|t_1|} + \frac{x}{|t_2|} + \frac{x-y}{x}\right) & \text{otherwise} \end{cases} \tag{9}$$

The selection of the Jaro-Winkler metric is justified by the fact that it computes the distance between two strings by comparing characters and symbols, as long as their corresponding positions. We consider this property as an interesting alternative to the token-based (or word-based) comparisons of the cosine and the Jaccard similarity metrics.

A. Effectiveness Evaluation

UMaP achieves product matching by generating clusters of same products. Therefore, to compare the output of UMaP with the true predicted matches, we applied the following methodology:

Initially, we iterate through each UMaP cluster and, for each product in the cluster, we create one pairwise match record with each of the rest of the products in the same cluster. In other words, we create a database with all the distinct product pairs (p_1, p_2) within a cluster. In the sequel, we perform the same procedure for the clusters of the dataset and we construct a second similar database. Finally, we compare the records of these two databases and attest the effectiveness of UMaP.

The matching quality was measured by employing the popular $F1$ score, given by the following formula:

K	α	$F1$	$Precision$	$Recall$
$K=3$	$\alpha=1$	0.32433	0.20470	**0.78032**
	$\alpha=2$	0.35216	0.22748	0.77929
	$\alpha=3$	0.33412	0.21313	0.77296
	$\alpha=4$	0.34597	0.22321	0.76880
	$\alpha=5$	0.33753	0.21637	0.76704
$K=4$	$\alpha=1$	**0.66370**	0.64175	0.68721
	$\alpha=2$	0.62118	0.60239	0.64118
	$\alpha=3$	0.61290	0.57920	0.65076
	$\alpha=4$	0.60302	0.56046	0.65258
	$\alpha=5$	0.58569	0.53552	0.64624
$K=5$	$\alpha=1$	0.48130	0.61997	0.39333
	$\alpha=2$	0.45771	0.59741	0.37096
	$\alpha=3$	0.43544	0.61239	0.33783
	$\alpha=4$	0.42029	0.57447	0.33136
	$\alpha=5$	0.40041	0.53044	0.32158
$K=6$	$\alpha=1$	0.35216	**0.71483**	0.23363
	$\alpha=2$	0.31339	0.69577	0.20225
	$\alpha=3$	0.29679	0.66443	0.19107
	$\alpha=4$	0.29022	0.62577	0.18892
	$\alpha=5$	0.27862	0.62301	0.17944

TABLE II
UMaP PERFORMANCE FOR VARIOUS VALUES OF K AND α.

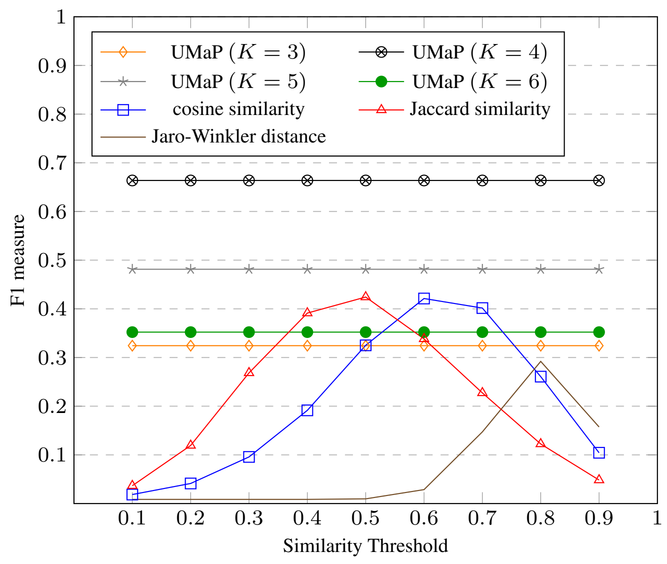

Fig. 2. Comparison of $F1$ scores for UMaP, cosine similarity, Jaccard similarity, and Jaro-Winkler distance

$$F1 = \frac{2PR}{P+R} \tag{10}$$

where P and R represent the Precision and the Recall values, respectively.

In the theoretical analysis of the algorithm, we introduced two tunable parameters:

i) K, which represents the maximum allowed length of a combination in words; and
ii) the constant $\alpha > 0$, which regulates the contribution of the average distance of a combination from the beginning of the titles.

On the other hand, the performance of the cosine similarity metric depends on a similarity threshold, which determines whether two entities match or not.

Initially, we examined the performance of UMaP for various values of K and α, with the aim of studying the behavior of the various parameters of the algorithm. In the three last columns of Table II, we record the measured values of $F1$, precision, and recall for the experiments that we conducted. Our method achieved the highest $F1$ score, approximately $F1 = 0.66$, by setting $K = 4$ and $\alpha = 1$. We will shortly show that this value was far superior to the $F1$ scores achieved by the adversary similarity metrics.

Moreover, the highest precision value was roughly 0.71 and was measured for $K = 6$ and $\alpha = 1$. Unfortunately, this setting was also accompanied by a low recall value (0.23) and consequently, this resulted in a low $F1$ score. On the contrary, the setting $K = 3, \alpha = 1$ produced the best recall value, about 0.78; however, the measured precision was also low in this case (approximately 0.20).

Regarding the variation of the K and α parameters, the examination of the results of Table II confirms two additional significant conclusions:

- Although the increase of the value of K leads to an improvement in the precision of the algorithm, it also reduces its recall. In other words, the usage of longer k-combinations does not enhance the performance of the algorithm in terms of $F1$ scores. The highest $F1$ scores were measured for $K = 4$, a setting which is also much more efficient, as we demonstrate in the following Subsection V-B.
- The distance of a k-combination from the beginning of the title indeed plays an important role in the solution of the examined problem. Equation 5 indicates that, by increasing the value of the constant α, we essentially weaken the impact of the distance. However, the experiments prove that such a weakening degrades the performance of the algorithm. Notice that the highest values of all evaluation metrics ($F1$, precision, and recall) were achieved for $\alpha = 1$.

Now let us compare the matching performance of our algorithm against the three aforementioned string similarity metrics. More specifically, we experiment with four UMaP settings ($K = 3, 4, 5$, and 6, with $\alpha = 1$) and 9 scenarios for each of the three examined string similarity measures, with similarity thresholds ranging from 0.1 to 0.9. The similarity threshold τ determines whether two entities e_1 and e_2 match or not. That is, e_1 matches e_2 if and only if their similarity value exceeds τ.

Figure 2 illustrates the variation of the $F1$ scores of COSim for different values of τ. Notice that UMaP is independent of this threshold; consequently, the plot of $F1$ against τ is a straight horizontal line in these cases. The highest $F1$ value achieved by COSim was about 0.42 and it was measured for a similarity threshold $\tau = 0.6$. At this point, the precision was equal to 0.35653, whereas the recall was 0.51457.

UMaP outperformed the highest $F1$ value of COSim in the cases of $K = 4$ ($F1 = 0.66$), and $K = 5$ ($F1 = 0.48$). The difference margins are remarkable: 36.3% for $K = 4$ and 12.5% for $K = 5$. On the other hand, COSim achieved better results than UMaP-$K = 3$ and UMaP-$K = 6$, within the range $\tau \in [0.6, 0.7]$. Consequently, UMaP performed very well for medium-sized k-combinations, that is, when these combinations contained at most 4 or 5 words. Beyond this length, the performance of our algorithm degrades.

Regarding the other two metrics of our evaluation, the settings $K = 4$ and $K = 5$ of UMaP achieved higher $F1$ values in all cases. More specifically, the Jaccard similarity metric performed equally well to cosine similarity. Its highest $F1$ measurement was also roughly 0.4242 and it was observed for a similarity threshold $\tau = 0.5$. At that point, the precision and the recall values were about 0.44080 and 0.40888, respectively. This value of $F1$ is about 36% lower than the 0.66 of UMaP $K = 4$. On the other hand, the $F1$ of the Jaro-Winkler distance metric was considerably lower compared to all other methods. Its greatest value was approximately 0.29 for $\tau = 0.8$, that is, more than 56% lower than the $F1$ of UMaP for $K = 4$. The measured precision and the recall were 0.24942 and 0.35226, respectively. Remarkably, the Jaro-Winkler metric was outperformed by all settings of UMaP (even by $K = 3$ and $K = 6$), and for all values of the similarity threshold τ.

Finally, notice that other works such as [1] and [2] report that cosine similarity achieved its best performance for other values of τ (more specifically, for $\tau = 0.3$ and $\tau = 0.4$). Moreover, for Jaccard similarity and Jaro-Winkler distance the maximum $F1$ was measured at $\tau = 0.5$ and $\tau = 0.8$, respectively. Therefore, it becomes clear that a complete and systematic experimental evaluation with these metrics, requires that the experiments should be repeated multiple times for all values of $\tau \in [0, 1]$.

B. Efficiency Evaluation

Undoubtedly, the extraction of combinations and permutations from a string is a computationally expensive task. In this Subsection we report some efficiency measurements which prove that UMaP is both viable and effective, even with large-scale datasets.

Apparently, the complexity of the algorithm primarily depends on the value of K. This parameter essentially determines the overall number of the k-combinations and permutations which will be produced during the execution. On the other hand, α is just a constant value in the denominator of equation 5 and does not affect the efficiency. Therefore, in the following discussion we focus on K only.

Table III records three execution statistics for various numbers of K in the range [2,6]. The first one, is recorded in the second column and represents the total running time of the algorithm. It includes all the steps described in the Algorithms 1 and 2 of Subsections IV-A and IV-B. Notice here, that, in our implementation, we employed standard data structures for the lexicon and the forward index and we did not applied any sophisticated optimization of the various parts of the algorithm. The other two statistics, recorded in the third and fourth columns of Table III, concern the total number of the computed k-combinations and permutations for each case.

	Durations (sec)	Combinations	Permutations
$K = 3$	1.55	2,896,310	3,292,384
$K = 4$	13.20	8,742,866	46,738,214
$K = 5$	292.32	21,733,514	1,177,518,713
$K = 6$	7177.36	46,482,486	$\simeq 1.3 \cdot 10^{12}$
COSim	171.47	–	–
JSim	244.89	–	–
JRD	282.30	–	–

TABLE III
EFFICIENCY EVALUATION OF UMAP AGAINST COSINE SIMILARITY, JACCARD SIMILARITY, AND JARO-WINKLER DISTANCE FOR VARIOUS VALUES OF K.

As anticipated, the results demonstrate that the running times grow exponentially with the increase in the value of K. For $K = 3$, the entire execution completes in almost 1 second and a half, but, for $K = 6$, the process consumes roughly 2 hours. The number of combinations and permutations also grow very fast with K; from a few millions for $K = 3$ to billions for $K = 6$. It seems that, for this particular dataset, a setting $K > 6$ is non-viable.

Nonetheless, in the previous discussion we showed that the increase of K actually leads to a decrease in the matching quality of UMaP. By comparing this outcome with the slow running times when $K \geq 6$, we can safely state that there is absolutely no reason in employing combinations which contain more than 6 words.

In addition, we demonstrated that the algorithm achieves the maximum $F1$ score by setting $K = 4$, followed by $K = 5$. The third row of Table III reveals that UMaP is pretty fast for $K = 4$; it processes the 16208 products of the dataset and clusters them in approximately 13.2 seconds. For $K = 5$, the execution is slower, but it is definitely within the acceptable limits; the algorithm consumes roughly 5 minutes in this case. Regarding the cases $K = 3$ and $K = 6$, the former is much faster than the latter. Since these two cases achieve similar $F1$ scores, $K = 3$ is more preferable than $K = 6$.

To provide a complete picture of the running times against the competitive string similarity metrics, in the three last rows of Table III we also record the time consumed by COSim, JSim and JRD to construct the matches among the 16208 products of the dataset. Of course, the adversary string similarity metrics do not operate by generating combinations and permutations, therefore, the values of the third and fourth column in these three last rows have been left blank intentionally. This part was also not optimized; we merely measure the duration of the pairwise comparisons and the computation of the required similarity values.

The results indicate that COSim required approximately 171.4 seconds to complete the matching task; in comparison to our proposed method, it was 13 times slower than UMaP-$K = 4$ and 1.7 times faster than UMaP-$K = 5$. Jaccard similarity was slower than COSim; apparently, the additional step of computing the unions in the denominator of eq. 8 resulted in this small delay. UMaP-$K = 4$ was approximately 18.5 times faster, whereas UMaP-$K = 5$ 1.2 times slower. The third of our examined metrics, JRD, was even slower; 21.4 times compared to UMaP-$K = 4$. The extra complexity of the calculation of the involved transpositions played a rather negative role.

The final outcome of the experimental evaluation of the proposed method against three of the most popular string similarity metrics, is that by setting $K = 4$ and $\alpha = 1$, we achieve: i) improved matching effectiveness by a significant margin of about 36%; and ii) much faster execution by a factor of at least x13.

VI. Conclusion

In this paper we introduced a new algorithm for matching product titles from different data feeds. This problem is particularly important in online shopping, since it allows the users to compare products (and especially their prices) from different electronic stores. The traditional string similarity metrics do not perform well in this area, since it is common that highly similar strings refer to different products, whereas other not similar strings describe the same product. The authors of [1] and [2] suggested the usage of Web search engines with the aim of enriching the product titles with several important missing words. Nevertheless, in case of large datasets, their approach is rather inefficient or even impossible, since the deployment of thousands of queries to a commercial search system is i) prohibitively expensive, and ii) forbidden by the terms of use of these systems.

On the contrary, our approach is based on the morphological analysis of the titles, and consists of two phases. During the first phase we process the titles and we construct combinations of K words. These combinations are stored within a lexicon data structure which additionally stores several statistics, such as the distance of the combination from the beginning of the title, and a frequency value which reflects the number of products than contain it. In case a new combination does not exist in the lexicon, we first compute all its permutations and we search the lexicon for each permutation. This strategy allows us to identify the same set of words, but in a different ordering. Moreover, we construct a typical forward index which for each product maintains a list of pointers to each combination. In the second phase, we use both the forward index and the statistics in the lexicon to assign a score to each combination. Finally, the combination which received the highest score is declared as the representative cluster label of a product.

The proposed algorithm was experimentally evaluated in terms of both effectiveness and efficiency against three traditional string similarity metrics, cosine similarity, Jaccard similarity, and Jaro-Winkler distance. The evaluation was performed by employing a dataset which we acquired by a crawling a popular product price comparison platform. The results lead to the conclusion that compared to the traditional similarity metrics, our approach achieves improved matching effectiveness by a significant margin of about 36%, whereas it is at least 13 times faster.

VII. Future Work

During the conduction of the experiments, we observed several remarkable issues regarding the behavior of our algorithm. These issues leave much room for future research and we firmly believe that by addressing them, we can further improve the effectiveness of our approach.

The first issue is about the phenomenon of a cluster 'split'. That is, a set of N identical products, sometimes gets incorrectly divided into two or more smaller subsets. By the time these lines are written, we are experimenting with a third processing phase for the algorithm. During this phase, we attempt to re-assign a cluster to the products which belong to such incorrectly divided subsets.

Another issue is to enhance the equation which assigns scores to the clusters, since equation 5 takes into consideration only global statistics for a combination. An interesting research line is to examine the possibility of utilizing additional local statistics, such as the word frequency or the word proximity within the titles of the products.

Finally, we will investigate the potential of improving the efficiency of the algorithm. There are several points to be examined, including: i) the data structures which accommodate the lexicon and the forward index; and ii) a robust blocking method which will allow us to terminate early the construction of the combinations and permutations.

References

[1] V. Gopalakrishnan, S. P. Iyengar, A. Madaan, R. Rastogi, and S. Sengamedu, "Matching Product Titles using Web-based Enrichment," in *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, 2012, pp. 605–614.

[2] N. Londhe, V. Gopalakrishnan, A. Zhang, H. Q. Ngo, and R. Srihari, "Matching Titles with Cross Title Web-search Enrichment and Community Detection," *Proceedings of the VLDB Endowment*, vol. 7, no. 12, pp. 1167–1178, 2014.

[3] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani, "Robust and Efficient Fuzzy Match for Online Data Cleaning," in *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, 2003, pp. 313–324.

[4] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios, "Duplicate Record Detection: a Survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, no. 1, pp. 1–16, 2007.

[5] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cabridge University Press, 2008.

[6] D. Bär, C. Biemann, I. Gurevych, and T. Zesch, "UKP: Computing Semantic Textual Similarity by Combining Multiple Content Similarity Measures," in *Proceedings of the First Joint Conference on Lexical and Computational Semantics-Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation*, 2012, pp. 435–440.

[7] A. Islam and D. Inkpen, "Semantic Text Similarity using Corpus-Based Word Similarity and String Similarity," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 2, no. 2, p. 10, 2008.

[8] J. Lu, C. Lin, W. Wang, C. Li, and H. Wang, "String Similarity Measures and Joins with Synonyms," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, 2013, pp. 373–384.

[9] W. H. Gomaa and A. A. Fahmy, "A Survey of Text Similarity Approaches," *International Journal of Computer Applications*, vol. 68, no. 13, 2013.

[10] C. F. Dorneles, R. Gonçalves, and R. dos Santos Mello, "Approximate Data Instance Matching: a Survey," *Knowledge and Information Systems*, vol. 27, no. 1, pp. 1–21, 2011.

[11] H. Köpcke, A. Thor, and E. Rahm, "Evaluation of Entity Resolution Approaches on Real-world Match Problems," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 484–493, 2010.

[12] H. Köpcke and E. Rahm, "Frameworks for Entity Matching: a Comparison," *Data & Knowledge Engineering*, vol. 69, no. 2, pp. 197–210, 2010.

[13] W. Shen, P. DeRose, L. Vu, A. Doan, and R. Ramakrishnan, "Source-Aware Entity Matching: a Compositional Approach," in *Proceedings of the 23rd IEEE International Conference on Data Engineering*, 2007, pp. 196–205.

[14] S. Cucerzan, "Large-scale Named Entity Disambiguation based on Wikipedia Data," in *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, 2007.

[15] M. Dredze, P. McNamee, D. Rao, A. Gerber, and T. Finin, "Entity Disambiguation for Knowledge Base Population," in *Proceedings of the 23rd International Conference on Computational Linguistics*, 2010, pp. 277–285.

[16] J. Hoffart, S. Seufert, D. B. Nguyen, M. Theobald, and G. Weikum, "KORE: Keyphrase Overlap Relatedness for Entity Disambiguation," in *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, 2012, pp. 545–554.

[17] M. Bilenko and R. J. Mooney, "Adaptive Duplicate Detection using Learnable String Similarity Measures," in *Proceedings of the ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2003, pp. 39–48.

[18] N. Jalbert and W. Weimer, "Automated Duplicate Detection for Bug Tracking Systems," in *Proccedings of the IEEE International Conference on Dependable Systems and Networks With FTCS and DCC*, 2008, pp. 52–61.

[19] C. Xiao, W. Wang, X. Lin, J. X. Yu, and G. Wang, "Efficient Similarity Joins for Near-duplicate Detection," *ACM Transactions on Database Systems*, vol. 36, no. 3, p. 15, 2011.

[20] H. Hajishirzi, W.-t. Yih, and A. Kolcz, "Adaptive Near-Duplicate Detection via Similarity Learning," in *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2010, pp. 419–426.

[21] I. Bhattacharya and L. Getoor, "A Latent Dirichlet Model for Unsupervised Entity Resolution," in *Proceedings of the 2006 SIAM International Conference on Data Mining*, 2006, pp. 47–58.

[22] P. Christen, "FEBRL: a Freely Available Record Linkage System with a Graphical User Interface," in *Proceedings of the 2nd Australasian Workshop on Health Data and Knowledge Management*, 2008, pp. 17–25.

[23] P. Jaccard, "The Distribution of the Flora in the Alpine Zone," *New phytologist*, vol. 11, no. 2, pp. 37–50, 1912.

[24] M. A. Jaro, "Advances in Record-Linkage Methodology as Applied to Matching the 1985 Census of Tampa, Florida," *Journal of the American Statistical Association*, vol. 84, no. 406, pp. 414–420, 1989.

[25] W. E. Winkler, "String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage," in *Proceedings of the Section on Survey Research Methods*, 1990, pp. 354–359.