Duplicate Detection in Web Shops using LSH to Reduce the Number of Computations

Iris van Dam
328808id@student.eur.nl

Gerhard van Ginkel
355735gg@student.eur.nl

Wim Kuipers
360436wk@student.eur.nl

Nikki Nijenhuis
353410nn@student.eur.nl

Damir Vandic
vandic@ese.eur.nl

Flavius Frasincar
frasincar@ese.eur.nl

Erasmus University Rotterdam, P.O. Box 1738, 3000 DR Rotterdam, the Netherlands

ABSTRACT

The amount of online shops is growing daily and many Web shops focus on the same product types, like consumer electronics. Since Web shops use different product representations, it is hard to compare products among different Web shops. Duplicate detection methods aim to solve this problem by identifying the same products in different Web shops. In this paper, we focus on reducing the computation time of a state-of-the-art duplicate detection algorithm. First, we construct uniform vector representations for the products. We use these vectors as input for a Locality Sensitive Hashing (LSH) algorithm, which pre-selects potential duplicates. Finally, duplicate products are found by applying the Multi-component Similarity Method (MSM). Compared to original MSM, the number of needed computations can be reduced by 95% with only a minor decrease by 9% in the F_1-measure.

CCS Concepts

•**Information systems** → **Entity resolution;** *Deduplication;* Clustering and classification;

Keywords

Duplicate detection; locality-sensitive hashing; Web shop products

1. INTRODUCTION

A wider range of products than ever before is available on the Web, due to the rapid growth of Web shops. Many Web shops sell the same products but use different representations for their products compared to other Web shops. It is possible that a Web shop provides additional information on a certain product which another Web shop does not provide, like customer ratings, accessories, or other product-related information, or uses different terminology to represent the same information.

SAC 2016, April 04 - 08, 2016, Pisa, Italy

© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-3739-7/16/04…$15.00

DOI: http://dx.doi.org/10.1145/2851613.2851861

It can be time consuming for customers to find the product that matches their requirements best, due to the different product representations. Also, it is impossible to compare all products from all Web shops manually. Web shop comparison sites, like kieskeurig.nl, have been developed to search the Web to find products that meet the requirements of a query. These sites aggregate data automatically from various Web shops. In this process it is necessary to perform product duplicate detection, to determine which Web shops offer the same product. Duplicate detection is not only used to find the duplicate products among different websites, but can also be applied to match a user query to products as close as possible. Fundamentally, duplicate detection comes down to the search for the nearest neighbors of products. However, duplicate detection is time consuming when all products have to be compared to each other, one by one, especially for large data sets. To decrease the computation time, we propose an approach based on Locality-Sensitive Hashing (LSH) [8] which reduces the number of comparisons. By using this technique for every product a set of nearest neighbors can be selected. To find duplicates of a product only the set of neighbors is considered, which leads to a reduction in the number of performed comparisons.

Applying LSH is not as easy as it may seem due to the use of different product representations across Web shops. The fact that not all Web shops provide the same information makes it necessary to first construct a general product representation which fits all products. For this general representation the model words of a product title are extracted. Model words are defined as words consisting of both numeric and non-numeric tokens. These model words are employed to produce a product binary representation that allows products to be compared to one another. Finally, we define the nearest neighbors as the products which have a similar binary representation.

The structure of the paper is as follows. In Section 2, we describe related work in the fields of LSH and duplicate detection. Then, in Section 3, we propose the general structure of our method for duplicate detection. Next, in Section 4, we give the general product representation using model words from the product title. Section 5 explains the used LSH and Section 6 contains the adjustments made to the Multi-component Similarity Method, a state-of-the-art method for product duplicate detection. We illustrate our method by means of an example in Section 7. Next, the results of our work are presented in Section 8. Last, Section 9 presents our conclusions and suggestions for further work.

2. RELATED WORK

In literature there are several duplicate and near-duplicate detection algorithms described, such as LSH and a blocking framework for Entity Resolution (ER). Entity resolution is the problem of defining and grouping different manifestations of the same real world object [6] which is similar to duplicate detection. LSH is used to find similar entities and has already been applied successfully for finding quickly nearest neighbors in large databases [12].

The topic of nearest neighbor search deals formally with a set S of n points in d dimensions. The goal is to build a data structure such that given a query point q, the point closest to q can be found quickly. A well-known problem in this field is the curse of dimensionality [3], i.e. the query search time usually grows rapidly in the dimension d.

In order to address this dimensionality problem, [8] proposes Locality-Sensitive Hashing (LSH). The key idea of LSH is to use hash functions such that the probability of collision is much higher for points that are close to each other than for those that are far apart. An advantage of this method is that the running times are much lower compared to brute-force approaches.

The amount of information on the Web has risen the last years, due to i.a. the increased possibilities to store large amounts of data on the Web combined with the possibilities of automatic extraction of information from raw data [11]. This effect has influence on the rise of highly heterogeneous information spaces (HHIS). Main characteristics of HHIS are non-structured data, high levels of noise, and large scale databases. In HHIS, ER has two forms: Dirty ER (input comprises a single entity collection) and Clean-Clean ER (the process of detecting duplicate pairs among two heterogeneous, duplicate-free, but overlapping collection of entities) [1, 5]. The data set used for this paper can be seen as a Clean-Clean ER since there are no duplicate products within a Web shop. [11] focuses on Clean-Clean ER. Data blocking is used in order to scale large volumes of data. Different blocking methods are discussed under which Token Blocking, Attribute Clustering Blocking, and Comparison Scheduling. The data used in this paper has similar characteristics to HHIS. We use an Attribute Clustering Blocking approach to exploit character patterns in the title attribute of the product description. We focus on title-based blocking method to detect duplicates due to the fact that titles provide a good summarization of a product and we want to keep processing times as low as possible.

The aim of this paper is to extend the Multi-component Similarity Method (MSM), which applies an adapted version of hierarchical single linkage clustering on a matrix containing dissimilarities between products [13]. Although MSM achieves a high F_1-measure, computation times are large. The calculation of the dissimilarity between two products takes a significant amount of time. Moreover, the dissimilarity between all pairs of products is calculated. Our paper aims to reduce the computation time of the MSM by reducing the number of pairwise comparisons.

In order to decrease the number of comparisons and thus the computation time LSH is applied as a pre-selection step. This is done using hash functions that divide all the products into several buckets. Ordinary hashing will obtain a good performance when products are exact semantic copies of each other [12]. LSH is more applicable to duplicate detection of Web products, since it searches for the n-nearest neighbors of a product. Due to the varying product representations products will differ too much among each other to use ordinary hashing, which makes it better to perform a local search within a set of candidate nearest neighbors. The working of LSH is described in many previous studies [2, 10, 12] and is similar to our LSH implementation described in Section 5.

In literature the evaluation of duplicate detection algorithms is twofold. In [11], the quality of blocks is evaluated in terms of *pair completeness* and *pair quality*. The first measures the fraction of duplicates sharing a block. The latter metric measures the average number of duplicates found per comparison performed. The quality of the duplicate detection algorithm overall is usually evaluated with the F_1-measure [4, 13].

3. METHOD OVERVIEW

As mentioned in the previous section, the approach proposed in this paper for product duplicate detection is based on extending MSM with LSH. Figure 1 gives an overview of our approach called Multi-component Similarity Method with Pre-selection (MSMP).

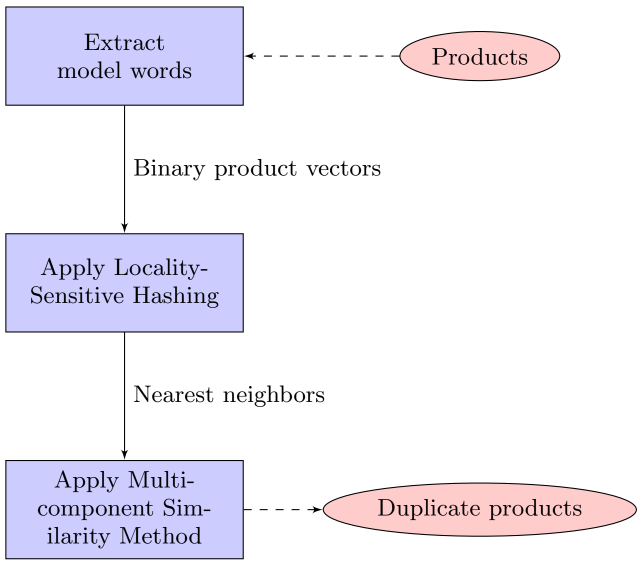

Figure 1: General overview of MSMP

For every product, the title is used to extract model words and obtain a binary vector representation of the product. The binary vectors are then compressed to a signature vector by min-hashing. These signature vectors are used to obtain the corresponding nearest neighbors using LSH. The nearest neighbors of a product, which is a subset of all products, are used to reduce the number of product comparisons in the final step of the algorithm. Each step of the procedure depicted in Figure 1 is discussed in more detail in the following sections.

4. EXTRACTING MODEL WORDS

The main idea of the method presented in this paper is to apply LSH as a pre-selection of products before using a clustering method to obtain duplicates. LSH can be applied to various forms of data. The title of a product often contains descriptive and distinctive information concerning the product. Therefore, information from the title, which is part of the product description, can be effectively used for hashing. In this paper, we will use model words from the *title* to create binary vectors representing the product based on the data from the title, in a similar way as in [13]. These binary vectors will be used to find the nearest neighbors by LSH. The formal definition of a model word can be expressed by a regular expression (regex). We use the following regex:

```
[a-zA-Z0-9]*(([0-9]+[ˆ0-9, ]+)|([ˆ0-9, ]+[0-9]+))[a-zA-Z0-9]*
```

The regex recognizes three different types of tokens: alphanumerical, numerical, and special characters. A model word contains at least two of these three types. Model words capture essential information about a product, e.g., `32"`, `720p`, `60Hz`, `32SL410U`, etc. Another advantage of using model words over regular words is that they are not appearing as often as regular words reducing the size of the representation scheme and thus the involved computation time.

We can formalize the procedure of obtaining binary vectors as follows. Let P be the set of product descriptions corresponding to the N products we consider in our data. Further, let $titleMW(p)$ denote the set of model words within the *title* attribute of product $p \in P$. The procedure of obtaining binary vectors is shown in Algorithm 1. In the first part we initialize MW as the set containing all model words from titles of all product descriptions. Secondly, for every product p we define a binary vector b^p by setting element i equal to 1 if the title of product p contains model word $i \in MW$.

Algorithm 1 Obtaining Binary Vectors

```
 1: $MW = \emptyset$
 2: for all products $p \in P$ do
 3:    for all model words $mw \in titleMW(p)$ do
 4:       $MW = MW \cup \{mw\}$
 5:    end for
 6: end for
 7: for all products $p$ in $P$ do
 8:    for all model words $mw \in MW$ do
 9:       if $mw \in title(p)$ then
10:          $b^p_{mw} = 1$
11:       else
12:          $b^p_{mw} = 0$
13:       end if
14:    end for
15: end for
16: return $b^p$ for all $p \in P$
```

By applying this algorithm, we have obtained a binary vector b^p for every product in our data set.

5. LOCALITY-SENSITIVE HASHING

In this section we explain how Locality-Sensitive Hashing (LSH) is used in order to reduce the computation time of finding product duplicates. LSH is applied on the resulting binary product vectors described in Section 4.

The main idea is to obtain a rough division of the products into different buckets, such that the clustering algorithm has to compare less products. The advantage of hashing is that it allows a fast mapping between products within a hash table. In this paper, LSH is used because of its ability to find nearest neighbors instead of exact duplicates, as explained in Section 2. Hence, we search for nearly similar products, in order to find true duplicates. LSH hashes two products to the same bucket if they are likely to be similar.

5.1 Min-hashing

Before LSH can be used effectively, the technique of minhashing is applied. This technique makes the product vectors more compact without losing too much of the original information that these vectors contain. Min-hashing is valuable because the original product vectors are long and sparse. They contain only 1 for the model words that are captured in the product title, and a zero for the remaining model words.

The min-hashing procedure is shown in Algorithm 2. The minhash of a binary vector is the number of the first row containing 1, after a random permutation of this binary vector is taken. Each time a minhash is determined, the minhash value is stored in the signature matrix.

Algorithm 2 Min-hashing

```
1: $n$ is the number of minhashes
2: $P$ is the set of all product vectors
3: S is an empty signature matrix of size $(n, |P|)$
4: for all $i = 1$ to $n$ do
5:    for all $v \in P$ do
6:       Determine for product $v$ under permutation $i$ the
         number of the row containing the first 1, $x$.
7:       $S(i, v) = x$
8:    end for
9: end for
```

Since we use a binary product representation, the Jaccard similarity between two product vectors allows us to measure how similar two products are. The cornerstone of the minhash approach is that the probability that two rows in the signature matrix are identical is equal to the Jaccard similarity of the binary product vectors. Therefore, if the signature matrix is not too small, it will give a shorter, representation of the original information. In our MSMP method, the number of permutations is fixed to a percentage of the original vector length. This is done in order to obtain a representative summary of each vector. Longer vectors, which contain a lot of information, get a larger signature matrix than vectors that contain less information. LSH can now be applied on the signature matrix instead of the product vectors.

5.2 Locality Sensitive Hashing

The general idea of LSH is to identify candidate pairs for which it must later be determined whether this pair is a pair of duplicates. The LSH method is depicted in Algorithm 3. LSH is applied on the signature matrix obtained in Section 5.1. This signature matrix is divided in b bands, each containing r rows. The choice of b and r must be such that the following equation holds:

$$n = r * b, \tag{1}$$

Here n is the length of the signature matrix. Each column, partitioned in b bands, of the signature matrix represents one product. For each band we use a hash function to hash the vector to a certain bucket. The used hash function hashes the vector to a bucket which number is defined by the sequence (string) of the elements (numbers) in the band. For example, when a band with four rows contains the values $[2, 3, 5, 8]$ it will hash the corresponding product to the bucket with number 2358. When two products hash to the same bucket for at least one band, the products are considered as a candidate pair. The hashing is done for all products and all b bands. Next, it is checked which products are candidate pairs using the hash table structure.

Algorithm 3 Locality-Sensitive Hashing

```
 1: $P$ is the set of all product vectors
 2: Divide the signature matrix M into $b$ bands, each containing $r$ rows
 3: for all $b$ bands do
 4:     Create an independent hash function $i$
 5:     for all $v \in P$ do
 6:         Hash $v$ to a bucket, based on the value of $i$ for the band
 7:     end for
 8: end for
 9: for all buckets do
10:     Label all pairs of products in this bucket as candidate neighbors
11: end for
```

Note that a higher value of b will give more false positives and a higher value of r will give more false negatives. Eventually the aim is to find a good division of b and r such that a good balance between the false positives and false negatives is created. A false positive combination means that two products are categorized as a candidate pair but they are not actually duplicates. A false negative is an actual duplicate pair that is not considered as a candidate pair.

In the MSMP method, it is important to lower the number of false positives, since they will undergo a time-consuming similarity check when the subsequent clustering is performed, as described in Section 6. On the other hand, the number of false negatives also must be small, since false negatives cannot be labeled as duplicates anymore by the clustering algorithm. The relation between false positives and false negatives can be represented by a threshold value t. The threshold value has the following relation with b and r:

$$\frac{1}{b}^{\frac{1}{r}} = t \tag{2}$$

Note that using Equations 1 and 2, the values of b and r are uniquely determined by the values of n and t. Thus, the only parameters for LSH are n and t. Two items are considered a candidate pair when they have at least one equal band hashing to the same bucket in the signature matrix. So, the choice of n and t directly influences the probability of two items becoming a candidate pair.

The products categorized as candidate neighbors are the pairs of products which are checked in the clustering algorithm described next in Section 6. The LSH algorithm has thus reduced the number of products which will be compared by the clustering algorithm.

6. MULTI-COMP. SIMILARITY METHOD

After using the LSH algorithm, as explained previously, we apply the Multi-component Similarity Method (MSM) as proposed in [13]. An important ingredient of this method is the similarity function, which calculates the similarity of two products.

The similarity function consists of three parts. First, the Key-Value Pairs (KVP) of products are compared. Q-grams are used to measure the similarity between keys. This measure uses tokens of q characters, in this case $q = 3$, taken from a sliding window from left to right of the strings containing the key. If the key similarity exceeds a certain threshold value, the q-gram similarity between the values is calculated and added with a certain weight to the final similarity of the two products.

Secondly, the KVP without a key-match are processed by the Hybrid Similarity Method (HSM) [4]. The model words from the values of these KVP are extracted and the percentage matching model words between the two products is calculated. The similarity based on this part of the method is also added with a certain weight to the final similarity of the two products.

The last part of the similarity measure is based on the Title Model Words Method [14] similarity making use of the model words present in the title, and also added with a certain weight to the final similarity between the two products. The weights of the three parts of the similarity measure are dependent on the minimum number of product features the two products contain and on whether products would be clustered when only the Title Model Words Method would have been used. The weighted sum of the similarity values of the three parts form the final similarity value of the two compared products.

MSM applies adapted hierarchical single linkage clustering on a matrix containing dissimilarities between products. This dissimilarity matrix is pre-processed by putting the dissimilarity values between two products from the same Web shop and/or with different brands on infinity. Whether two products have the same brand, is checked using a list of television brands obtained from the Web. The remaining entries in the dissimilarity matrix are calculated by using the previously described similarity function for each pair of products. However, our approach determines the dissimilarity matrix in an adapted way. The distance between two products is not only set to infinity if the products are from the same Web shop or have different brands, but also when they are not assigned as a candidate pair by the LSH method. By this adaptation, the number of comparisons and thus the computation time will decrease.

Adapted hierarchical single linkage clustering is applied on the previously obtained dissimilarity matrix. The distance between two clusters is given by the shortest distance between a pair of products from these clusters. However, if there is a pair of products with a distance of infinity, the distance between the two clusters is also infinity. Iteratively, the two nearest clusters are merged until the smallest cluster distance exceeds a predefined threshold. Products in the same clusters are considered duplicates.

7. IMPLEMENTATION EXAMPLE

To illustrate our method described in the previous sections, we use an example consisting of five products. The products are described by means of a KVP representation. Every product description contains, among other values, an attribute for *title*. As mentioned in Section 4 the binary vectors representing the products are obtained by using model words from the *title* attribute. The titles of the products in our example are listed in Table 1. The product combinations (2,3) and (4,5) are in fact duplicates. We will apply the steps discussed in Sects. 3-6 to this example.

Product ID	Title	Web shop
1	SuperSonic 32" 720p LED HDTV SC-3211	Newegg
2	Samsung UN46ES6580 46-Inch 1080p 120Hz 3D HDTV	Amazon
3	Samsung 46" 1080p 240Hz LED HDTV UN46ES6580	Newegg
4	Toshiba - 32" / LED / 720p / 60Hz / HDTV	Bestbuy
5	Toshiba 32" 720p 60Hz LED-LCD HDTV 32SL410U	Newegg

Table 1: Example of five products having different title attributes

In the first part of the algorithm, the model words are extracted from the *title* attributes and a binary vector is obtained as a product representation for every product. The procedure as described in Section 4 results in the binary vectors of Table 2.

	Product ID				
Model word	1	2	3	4	5
720p	1	0	0	1	1
1080p	0	1	1	0	0
60Hz	0	0	0	1	1
120Hz	0	1	0	0	0
240Hz	0	0	1	0	0
3D	0	1	0	0	0
SC-3211	1	0	0	0	0
UN46ES6580	0	1	1	0	0
46-inch	0	1	0	0	0
32L410U	0	0	0	0	1

Table 2: Example of binary vector product representations

We have obtained binary vectors for all products. These vectors represent the model words present in the title of each product. The length of the binary vectors is 10 in this example. In case the number of available products is high, the length of the vectors will be large since more model words appear in the titles.

The size of the previously obtained sparse vectors can be reduced by compressing the binary vectors to signature vectors using min-hashing. We now compress the binary vectors in the example to signatures of length $n = 4$. Hence, we need 4 permutations p_i of the 10 rows of the binary vectors:

$$p_1 = [1, 6, 2, 9, 3, 10, 8, 4, 5, 7]$$
$$p_2 = [6, 1, 5, 9, 10, 3, 7, 2, 8, 4]$$
$$p_3 = [5, 7, 8, 3, 6, 1, 2, 10, 4, 9]$$
$$p_4 = [2, 7, 4, 3, 9, 8, 10, 5, 6, 1]$$

The permutations p_1-p_4 as shown above will be used to obtain the signature vectors for products 1-5. These permutation have been generated randomly. The retrieved signature vectors for the products are listed in Table 3.

These signature vectors are the input vectors for the LSH algorithm. For this example we fix the number of bands b at 4. Consequentially the number of rows r within a band is equal to 1, since the relation $n = b * r$ should hold and n is equal to 4. Products i and j will be classified as neighbors by LSH if the signature vectors are equal in at least 1 band. Since $r = 1$, we need the signature vectors to have at least 1 row with equal entries in order to consider the corresponding products as candidate pair. In the example, products 1 and 2 are not neighbors, since they have different entries in all rows. Products 2 and 3 will be classified as neighbors because they have the same value in permutation 4. The neighbors obtained after LSH are given in Table 4. If products i and j are neighbors, the element of the matrix in Table 4 is equal to 1. We assume products are not neighbors with themselves.

	Product ID				
Permutation	1	2	3	4	5
p_1	1	2	3	1	1
p_2	2	1	3	2	2
p_3	2	3	1	4	4
p_4	2	1	1	4	4

Table 3: Example of signature matrix product representations

	Product j				
Product i	1	2	3	4	5
1	0	0	0	1	1
2		0	1	0	0
3			0	0	0
4				0	1
5					0

Table 4: Example of potential product duplicates

Recall from the beginning of this section that product combinations $(2, 3)$ and $(4, 5)$ are in fact duplicates. LSH has classified these pairs as neighbors.

Further, one can see that the number of product combinations classified as neighbors is 4. This is smaller than 10, the maximum number of combinations with 5 products. After applying LSH we use the MSM method as described in Section 6. Combinations of products not classified as neighbors will not be compared and their dissimilarity will be set to infinity. Because only 4 out of 10 combinations remain after LSH, we have reduced the number of comparisons, for this case, by 60% without losing on performance for duplicate detection, since all true duplicates will still be considered.

8. EVALUATION

In this section, the results of our proposed MSMP are evaluated. To assess the performance of our method, we use it for duplicate detection on a data set of TV's, obtained from four different Web shops: Amazon.com, Newegg.com, BestBuy.com, and TheNerds.net. The data of these shops consists of 163, 668, 773, and 20 TV's, respectively. This gives a total of 1,624 TV's, of which 1,262 are unique.

In order to analyze the performance of the methods described in this paper we apply them to the data as mentioned above. For stable results we use bootstrapping and evaluate performance on every bootstrap. A bootstrap is a random selection from the data with replacement. The size of bootstraps used is around $60 - 65\%$ of the data, so every bootstrap contains around $1,000$ products. The performance measures are computed as the average performance over all bootstraps.

The main contribution of this paper is the pre-selection of products before applying clustering. Therefore, we first analyze the quality of the pre-selection of the products to be clustered, which is performed by the LSH part in the MSMP. A detailed description of this part of the methodology was given in Section 5. Secondly, we analyze the overall performance of MSMP.

Performance LSH

The following setup is used to evaluate the performance of the pre-selection of duplicates by the LSH method.

First, for a fixed input parameter combination, the MSMP method is run for 100 bootstraps, only now with a modified perfect similarity function. This function assigns similarity equal to 1 if two products are in the same bucket and are the same according to the golden standard, and 0 in all other cases. The obtained similarities are used for clustering as described in Section 6. By using a modified perfect similarity measure instead of the default similarity measure in the MSMP, we are able to isolate the effect of pre-selection by the LSH method on the end result.

The performance of the LSH method depends on the quality of the product vectors, the size n of the signature matrix, and the threshold value t. The size of the signature matrix has no large impact on the total running time of the clustering algorithm, since computing similarities with MSMP consumes the most time by far. Since a high value of n will give more reliable and stable results, we choose n to be 50% of the product vector length.

The performance of the LSH method, and hence the blocks, are evaluated in terms of pair completeness and pair quality. Because we use the modified perfect similarity function the precision is always equal to 1. Therefore we evaluate the recall, or pair completeness. This measures the duplicates found as a fraction of the total number of duplicates. The pair quality is a measure for the average number of duplicates found per comparison. Hence, it measures the efficiency of a comparison. Note that the pair quality takes values on the interval $[0, 2]$ since 1 comparison can lead to at most 2 found duplicates.

We evaluate the pair completeness and the pair quality for different threshold values t. A higher threshold value will lead to less comparisons since less products will be considered candidate pairs. We are interested in reducing the running time by reducing the number of comparisons. Since the number of comparisons has a direct effect on the running time, we plot the performance measures against the number of performed comparisons with respect to the total number of possible comparisons.

More comparisons lead to a higher running time but also to a higher pair completeness. We analyze the characteristics of this trade-off. Therefore the algorithm is executed for 100 bootstraps for a grid of threshold values t, varying from 0.05 to 0.95 with a step size of 0.05. All results are averaged over the bootstraps. The number of pairwise comparisons is given as a fraction of the pairwise comparisons in MSM.

When all comparisons are performed, no information is lost and the recall is equal to 1. Reducing the number of comparisons causes our algorithm to loose the ability to find some duplicates, which deteriorates the performance. One can note based on Figure 2, that a pair completeness of 0.533 is already achieved when the number of pairwise comparisons is reduced with more than 97% with respect to the MSM. Performing 18.3% of the pairwise comparisons gives a recall of 0.800 and when 74.4% of the comparisons are performed, little information is lost about the duplicates, since a pair completeness of 0.955 is achieved.

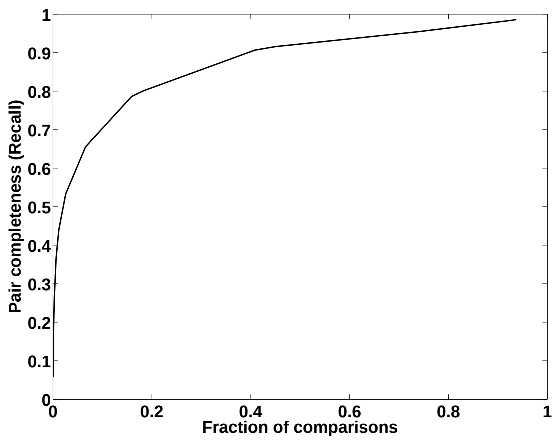

Figure 2: Pair completeness versus fraction of comparisons

We observe that performing more comparisons leads to more information about possible duplicates. However, the information obtained per comparison may decrease. We analyze the characteristics of this trade-off by measuring the pair quality. Figure 3 shows this trade-off for the exact same dataset used for Figure 2.

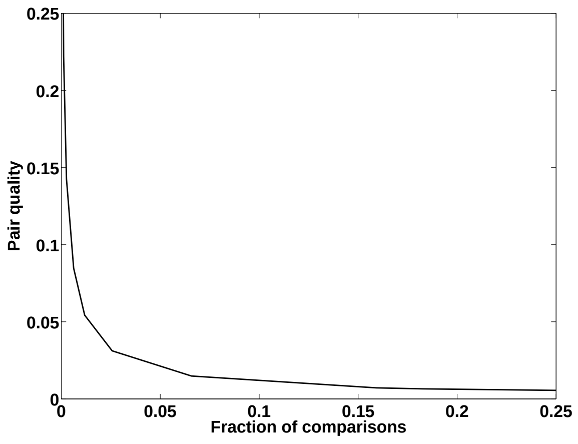

Figure 3: Fraction of lost duplicates versus fraction of comparisons

In Figure 3 we observe a rapid increase in the pair quality when the fraction of comparisons approaches 0. This is caused by the fact that our data set contains a relatively small amount of duplicates. Therefore, a high pair quality is observed only if very little comparisons are performed. LSH is quite effective: when 0.1% of the comparisons are performed, on average a duplicate is found every 4 comparisons. The pair completeness for 0.1% of the comparisons is 0.44, which indicates that 44% of the duplicates are discovered in this case.

To summarize, a significant increase in pair quality is achieved by performing very little comparisons. Furthermore, we observe that with respect to MSM a significant reduction in the number of pairwise comparisons and thus the computation time can be achieved, while keeping the deterioration of the recall small.

Performance MSMP

Below we compare our results with the results of MSM [13]. The performance is evaluated with the F_1-measure, which is the harmonic mean of precision and recall. We run MSMP for different fixed threshold values t. In every run, consisting of 100 bootstraps, the parameters of the MSM clustering and similarity methods are optimized over a grid of parameters. The best performing set of parameters is chosen for every value of t. Furthermore, the average number of comparisons is stored for every threshold t.

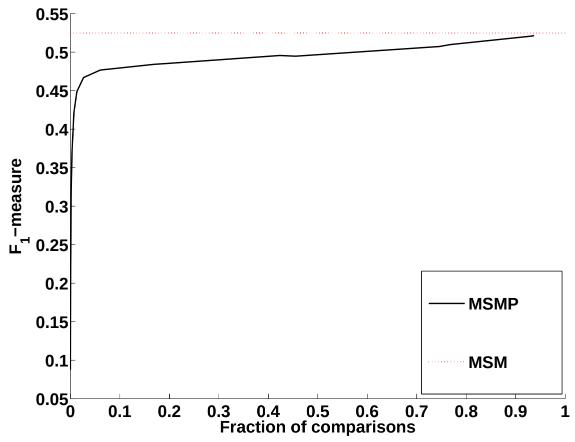

Figure 4: F_1-measure of MSMP compared to the benchmark for different fractions of comparisons

In Figure 4 the performance of the MSMP method proposed in this paper is evaluated in terms of the F_1-measure. By LSH we have reduced the number of pairwise product comparisons with respect to the benchmark we consider, the MSM method. All results are averages over 100 bootstraps. The F_1-measure of our benchmark, which is 0.525, is also computed as an average over 100 bootstraps. It is depicted by the red dashed line in Figure 4. Note that for the benchmark all comparisons are performed, thus the fraction of comparisons is equal to 1.

We observe that a reduction in comparisons achieved by LSH leads to a lower F_1-measure. However, a large decrease in computation time leads to a minor decrease in performance. This trade-off between computation time and F_1-measure is approximately linear up to a reduction of 95% of the comparisons. Consequently, by performing only 5% of the comparisons MSMP still achieves, on average, an F_1-measure of 0.477.

We can conclude that for the given product data LSH is a very powerful method, able to significantly reduce computations whilst the performance of the target, namely finding duplicates, is still considerably high. Hence, applying LSH to model words from the title proves to be an effective way to reduce computation time.

9. CONCLUSIONS

In this paper, we gave a solution to the scalability problem of product duplicate detection in Web shops. Our proposed method builds on the Multi-component Similarity Method (MSM) [13], a state-of-the-art product duplicate detection method. We proposed an LSH-based filtering step to lower the number of necessary comparisons for duplicate detection by performing a pre-selection of candidate pairs.

First, we described all products by means of a representation based on model words in the title. Each representation contains a set of model words that is used to obtain binary vectors. The length of the binary vectors depends on the number of model words that appear in all titles of the products. This length can be reduced by compressing the binary vectors to signature vectors by means of min-hashing. After min-hashing, the obtained signature vectors are the input for the LSH algorithm. The vectors are divided in b bands of r rows and two products are classified neighbors if the signature vectors are equal in at least 1 band. Finally, we apply a twofold analysis. First, solely the LSH part of the MSMP is evaluated, using a perfect similarity function for MSM. Second, the overall performance of MSMP is evaluated and compared with MSM.

Our Locality-Sensitive Hashing algorithm reduces the number of comparisons for MSM by pre-selecting products and only comparing those products that are classified as neighbors. This reduces the running time of MSM, since less time-consuming computations of similarity values are required.

Our results show that a significant reduction in computation time can be achieved by using LSH, at the cost of only a relative small decrease in pair completeness. For example, more than 80% of the duplicates can be found with a perfect clustering algorithm doing only 18% of all possible comparisons.

If we use MSM in combination with LSH we can reduce the number of needed comparisons and thus the computation time drastically while keeping the performance in terms of F_1-measure relatively high. For example, with 5% of the comparisons we achieved an F_1-measure of 0.477, which is below to the F_1-measure of 0.525 of MSM. Hence, we showed that our proposed method successfully addresses the scalability issue of product duplicate detection on the Web.

As future work we would like to improve the product vector representations that are based on more information than solely the model words from the title, using key-value pairs from product descriptions and thus provide better input for the Locality Sensitive Hashing method. Also, we would like to experiment with other blocking methods, e.g., by making use of not only words but also q-grams and combinations of these. In addition, other clustering algorithms, e.g., modularity-based clustering or spectral clustering, are planned to be investigated in the context of product duplicate detection on the Web. Last, we would like to employ map-reduce implementations of the proposed algorithms, i.e., blocking [7], computing dissimilar values for candidate pairs, and clustering [9], in order to further increase the efficiency of our solution.

Acknowledgments

The authors of this paper are partially supported by the Netherlands Organisation for Scientific Research (NWO), under the Mozaiek project *'Semantic Web Enhanced Product Search (SWEPS)'* (project 017.007.142), and the Dutch national program COMMIT. We would like to thank Jantijn Kromwijk and Maarten Lemken for many fruitful discussions and their help in gathering the data set used in the experiments. Further, we thank SARA Computing and Networking Services for their support in using the Lisa Computer Cluster.

10. REFERENCES

[1] P. Christen. A survey of indexing techniques for scalable record linkage and deduplication. *IEEE Transactions on Knowledge and Data Engineering*, 24(9):1537–1555, 2012.

[2] O. Chum, J. Philbin, and A. Zisserman. Near duplicate image detection: Min-Hash and TF-IDF weighting. In *19th British Machine Vision Conference*. British Machine Vision Association, 2008. http://www.bmva.org/bmvc/2008/papers/119.pdf.

[3] K. L. Clarkson. An algorithm for approximate closest-point queries. In *Proceedings of the Tenth Annual Symposium on Computational Geometry*, pages 160–164. ACM, 1994.

[4] M. de Bakker, F. Frasincar, and D. Vandic. A hybrid model words-driven approach for web product duplicate detection. In *25th International Conference on Advanced Information Systems Engineering*, volume 7908, pages 149–161. Springer, 2013.

[5] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 19(1):1–16, 2007.

[6] L. Getoor and A. Machanavajjhala. Entity resolution: Tutorial. http://www.umiacs.umd.edu/~getoor/Tutorials/ER_VLDB2012.pdf, 2012.

[7] S.-C. Hsueh, M.-Y. Lin, and Y.-C. Chiu. A load-balanced mapreduce algorithm for blocking-based entity-resolution with multiple keys. In *Twelfth Australasian Symposium on Parallel and Distributed Computing*, volume 152. Australian Computer Society, 2014.

[8] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *30th Annual ACM Symposium on Theory of Computing*, pages 604–613. ACM, 1998.

[9] C. Jin, M. M. A. Patwary, A. Agrawal, W.Hendrix, W. k. Liao, and A. Choudhary. Disc: A distributed single-linkage hierarchical clustering algorithm using mapreduce. In *4th International SC Workshop on Data Intensive Computing in the Clouds*, 2013.

[10] Y. Ke, R. Sukthankar, and L. Huston. Efficient near-duplicate detection and sub-image retrieval. In *12th ACM International Conference on Multimedia*, pages 869–876. ACM, 2004.

[11] G. Papadakis, E. Ioannou, T. Palpanas, C. Niederee, and W. Nejdl. A blocking framework for entity resolution in highly heterogeneous information spaces. *IEEE Transactions on Knowledge and Data Engineering*, 25(12):2655–2682, 2013.

[12] M. Slaney and M. Casey. Locality-sensitive hashing for finding nearest neighbors. *IEEE Signal Processing Magazine*, 25(2):128–131, 2008.

[13] R. van Bezu, S. Borst, R. Rijkse, J. Verhagen, F. Frasincar, and D. Vandic. Multi-component similarity method for web product duplicate detection. In *30th Annual Symposium on Applied Computing*, pages 761–768. ACM, 2015.

[14] D. Vandic, J.-W. Van Dam, and F. Frasincar. Faceted product search powered by the Semantic Web. *Decision Support Systems*, 53(3):425–437, 2012.