Near-Duplicate Video Detection Based on an Approximate Similarity Self-Join Strategy

Henrique Batista da Silva, Zenilton Patrocino Jr., Guillaume Gravier, Laurent Amsaleg, Arnaldo de A. Araújo, Silvio Jamil F. Guimarães

Near-Duplicate Video Detection Based on an Approximate Similarity Self-Join Strategy

Henrique B. da Silva*, Zenilton K. G. do Patrocínio Jr.‡, Guillaume Gravier†, Laurent Amsaleg†, Arnaldo de A. Araújo* and Silvio Jamil F. Guimarães‡
*Department of Computer Science, Universidade Federal de Minas Gerais, Belo Horizonte, MG, Brazil
†IRISA (CNRS, Univ. Rennes 1), Campus de Beaulieu, Rennes, France
‡Instituto de Ciências Exatas e Informática, Pontifícia Universidade Católica de Minas Gerais, Belo Horizonte, MG, Brazil
{henrique.silva, arnaldo}@dcc.ufmg.br, {guillaume.gravier, laurent.amsaleg}@irisa.fr, {zenilton, sjamil}@pucminas.br

***Abstract*—The huge amount of redundant multimedia data, like video, has become a problem in terms of both space and copyright. Usually, the methods for identifying near-duplicate videos are neither adequate nor scalable to find pairs of similar videos. Similarity self-join operation could be an alternative to solve this problem in which all similar pairs of elements from a video dataset are retrieved. Nonetheless, methods for similarity self-join have poor performance when applied to high-dimensional data. In this work, we propose a new approximate method to compute similarity self-join in sub-quadratic time in order to solve the near-duplicate video detection problem. Our strategy is based on clustering techniques to find out groups of videos which are similar to each other.**

I. Introduction

In recent years, research involving video retrieval has been increased, mainly due to the popularity of sites for video sharing and viewing over the Web, e.g., YouTube. However, according to [1], several videos are either identical or almost identical when they are compared to each other. Usually, they differ in the format, encoding parameters, and the use of edition operations for the addition and/or removal of frames. This kind of videos was defined by [2] as near-duplicate as their visual content is similar. As reported by [3], in a query for videos done in the Web, the search results presented 27% of redundancy and 20% of exact duplicate videos among all near-duplicate videos.

There are many works addressing the problem of Near Duplicate Video Retrieval (NDVR) and Near Duplicate Video Localization (NDVL). According to [4], NDVR and NDVL can be classified into: (i) frame-level approaches, that figure out video similarity by computing similarity frame-by-frame from the videos; (ii) spatiotemporal methods using representative segments from videos based on spatiotemporal features [4], [5]; (iii) video-level approaches, in which videos are represented by a global signature; and (iv) hybrid hierarchical methods, that perform a filter-and-refine approach, in which, non-near duplicate videos are filtered out (filtering step) and a ranking of the remain videos is performed (refine step) [3]. Some methods are also based on hash approaches to match similar videos and solve NDVR problem using global or local visual features as video representation [6]. As the amount of data grows, some works address the near-duplicate video detection problem by using parallel solutions such as MapReduce framework or GPU for large-scale NDVR problem [1]. When using the video-level strategy, the common approach to identify near-duplicate videos is based on the following: (i) videos are segmented into shots that are represented by key-frames; (ii) each key-frame is described by a signature (or descriptor), usually belonging to a high-dimensional space; (iii) then a global signature is computed over the key-frame descriptors to represent the whole video; and (iv) the similarity among videos can be computed by using either the set of signatures of key-frames or the global signature.

Unfortunately, the common strategy is not efficient for finding near-duplicate objects in a huge amount of high-dimensional multimedia data since it is necessary to compute the similarity between each pair of elements. The retrieval of all similar pairs of videos from two collections is called similarity join, hereafter. More formally, let R and S be two video collections. The similarity join between R and S is the set of all pairs of elements $r \in R$ and $s \in S$ such that $d(r, s) \leq \lambda$, in which $d(r, s)$ and λ are a distance function and a given threshold, respectively. If both video collections are the same, this operation is called similarity self-join.

From an application point of view, similarity join has been adopted to find pairwise elements in two datasets, for instance: (i) similar pairs of strings from two collections of strings which is an important operation in data integration and cleansing [7]; (ii) near-duplicate objects (regarding Web pages documents) [8]; (iii) data groups using density-based clustering [9]; and (iv) similar time-sequence data [10]. To find near-duplicate objects by similarity join operation, a naive (or standard) method computes a distance between each pair of objects in $O(n^2)$ time, where n is the size of the dataset [11]. Unfortunately, this computation is too expensive for many applications. In order to reduce the computational effort of similarity join, many approaches have been proposed [10], [11], [12], [13], [14]. Moreover, it is also desirable to avoid the output explosion problem by obtaining all groups of elements that are similar to each other in the output instead explicitly enumerating all qualifying pairs, as [11] does.

Thereby, due to limitations imposed by quadratic compu-

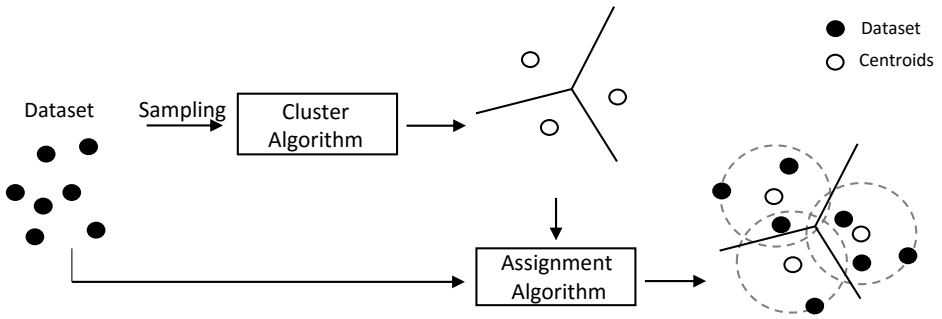

Fig. 1. Outline of our method for computing an approximate similarity self-join.

tational time to perform similarity join operation, the use of an approximate algorithm is a promising direction and it has attracted considerable attention. Even though many approaches have been proposed to reduce the computational effort to solve similarity join, high-dimensional data are still hard to manipulate due to the *curse of dimensionality*. Thus, in this work, we address near-duplicate video detection problem by using approximate similarity self-join operation to find groups of videos which are similar to each other in a database. In Figure 1, we outline our proposed method. In a general way, as the retrieved elements in each cluster are similar to each other, we can consider them as near-duplicate elements. It is important to note that we applied our method to a known dataset used to identify near-duplicate videos. We also applied our strategy to a dataset containing SIFT points in order to evaluate the performance of our strategy in a larger database.

As video descriptor, we adopted a high-dimensional global signature (with thousands of dimensions) based on BossaNova Video Descriptor (BNVD), introduced by [15]. To describe video content, first we have used SIFT descriptors that were combined into BossaNova, a mid-level image representation [16]. Finally, BNVD is obtained by applying max function on BossaNova vectors [15].

The rest of this paper is organized as follows. In Section II, we review some related work. In Section III, we present the proposed approach to solve the near-duplicate video detection problem. In Section IV, we present the results of the experiments performed. Finally, in Section V, we draw some conclusions and discuss future work.

II. Similarity Self-Join

The main goal of the similarity self-join operation is to retrieve all pairs of similar elements belonging to a dataset. Regarding similarity self-join, the first kind of strategy for avoiding the enumeration of all pairs is called *filter-and-verification approach*. This strategy is based on the identification of candidate pairs followed by the verification of those pairs. The identification of candidate pairs is used to filter out pairs which are not in the solution space in order to decrease the number of pairs that need to be verified. This strategy was successfully used by [7], [14] to cope with similarity self-join in the string domain. Several recent approaches, such as *qChunk*, *PPJoin*, *Ed-Join*, *adaptJoin*, developed strategies for improving the filter step in order to achieve efficiency on the search process. These algorithms were exhaustively compared in [7]. According to [14], parallel approaches by using MapReduce framework have widely been used in large datasets in order to reduce the computation time needed to perform a similarity (self-)join. For more information, the reader should refer to [9], [17], [18].

Despite the simplicity of the filter-and-verification approach, its main drawback is the difficulty to produce high performance when applied to large datasets and to high-dimensional spaces. To cope with these issues, some authors have proposed the computation of similarity join directly by methods based on tries [19]. Even if the filter-and-verification approach and trie-based methods work well for small datasets, they still have poor performance on large datasets in high-dimensional spaces since the number of distance computation is very high. In this case, there are many approaches that try to cope with high-dimensional data using hash-based techniques. For instance, the method proposed in [20], called *Sim-min-Hash*, a generalization of min-Hash, solves image search and link problem handling billions of high-dimensional local descriptors in a fast way. In most cases, high-dimensional data has been handled by application of *Locality Sensitive Hashing* (LSH), due to its suitable performance on high-dimensional space, as was presented in [21], a Bayesian algorithm, called BayesLSH, for performing candidate pruning and similarity estimation by LSH. Additionally, another method based on hash, proposed in [22], called *Data Sensitive Hashing* (DSH), solves k-NN problem by improving hashing family through the combination of adaptive boosting and spectral techniques to hash, with large probability, k-NN pairs together.

Accordingly, there are many applications to similarity join using high-dimensional data, like near-duplicate video detection, for instance. However, there are several challenges related to the similarity join problem. Many different types of data are represented in high dimensional space, generally, using in many cases hundred or thousand dimensions. Furthermore, many applications have to deal with a huge amount of data. Similarity join in large datasets has become a difficulty task.

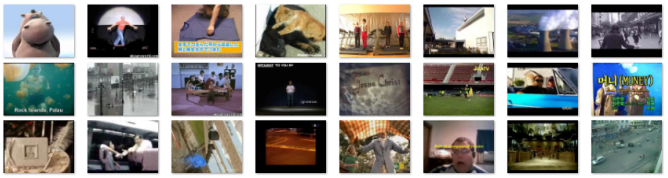

Fig. 2. CC_WEB_VIDEO Dataset.

Thus, another promising direction to solve similarity join problem is the approximate approach. Approximate algorithm can solve similarity join problem missing some results. In this paper, we aim to solve the near-duplicate video detection problem using an approximate similarity self-join approach based on a clustering algorithm.

III. Similarity Self-Join for Near Duplicate Video Detection

A. BossaNova Video Descriptor (BNVD)

Let $\mathcal{V}$ be a video sequence. $\mathcal{V} = \{f^i\}$, $i \in [1, N]$, where f^i is the keyframe[1] of the shot i and N is the number of keyframes. Let $\mathbb{Z}$ be a set of BossaNova vectors computed for the video $\mathcal{V}$. $\mathbb{Z} = \{\mathbf{z}^i\}$, $i \in [1, N]$, where $\mathbf{z}^i$ is a BossaNova vector extracted for the keyframe f^i. The BNVD of $\mathbb{Z}$ can be modeled by a function h, according to [15], as follows:

$$\begin{aligned} h : \mathbb{R}^Z &\longrightarrow \mathbb{R}^Z, \\ \mathbb{Z} &\longrightarrow h(\{\mathbf{z}^i\}) = [[o_{m,b}], p_m]^{\mathrm{T}}, \\ o_{m,b} &= \text{median}(z^i_{m,b}), \\ p_m &= \text{median}(t^i_m), \end{aligned} \tag{1}$$

where $Z \subset \{1, ..., M\} \times \{1, ..., B+1\}$, and $\mathbf{z}^i = \left[\left[z^i_{m,b}\right], t^i_m\right]^{\mathrm{T}}$, M is the number of visual codewords, and B denotes the number of bins of each BossaNova histogram.

B. Near Duplicate Video Detection

Let us remember the definition of the similarity join operation. Let R and S be two datasets containing n and m elements, respectively. The result of the similarity join between R and S is the set of all pairs of elements $r \in R$ and $s \in S$ such that $d(r, s) \leq \lambda$, in which $d(r, s)$ and λ are a distance function and a given threshold, respectively. If both datasets are the same, this operation is called similarity self-join. The computational cost of the naive algorithm to compute this operation is $O(n^2)$. Even if this cost can be considered reasonable, it is unfeasible for huge datasets. Thus, in this work, we are interested in computing a similarity self-join by using a sub-quadratic method trying to maintain the same quality measures, in terms of precision and recall, of the exact solution which runs in a quadratic time.

In order to decrease the computational time, it is mandatory to avoid distance computations between all pairs. For that, our method for approximate similarity self-join can be divided in two main tasks, as illustrated in Figure 1: (i) the clustering; and (ii) the bucket assignment. The clustering step aims at identifying salient elements to well represent all dataset, and then, we assign similar elements to the same bucket in which all elements could be considered as similar to each other. However, before applying the proposed similarity self-join strategy to near-duplicate video detection, the videos must be transformed into only one feature vector, and in this work, we have used BossaNova mid-level representation (see [16] for more information) for describing the video.

[1]A keyframe is a frame that represents the content of a logical video unit, like a shot or scene, for example.

Algorithm 1 Approximate similarity self-join algorithm: the assigment task

Require: $\mathbf{p} = \{p_1, \ldots, p_N\}$ ▷ Data points, $p_i \in \mathbb{R}^d$
Require: $\mathbf{c} = \{c_1, \ldots, c_K\}$ ▷ Centroids, $c_j \in \mathbb{R}^d$
Ensure: $\mathbf{B} = \{B_1, \ldots, B_K\}$ ▷ Buckets, $B_j \subseteq \mathbb{R}^{N \times d}$

```
// Initialization of a bucket B_i for each cluster
for j = 1 to K do
    B_j ← ∅
end for
// Assign each point to at most m buckets and fill in buckets
for i = 1 to n do
    for j = 1 to K do                          ▷ Fill in similarity array s
        s_j ← (j, d(p_i, c_j))
    end for
    s ← topK(s,m)                              ▷ Top-m highest similar centroids
    for k = 1 to m do
        if isViable(B,score(s_k),type) then
            // Function index returns index of pair in s
            ĵ ← index(s_k)
            // Add point to bucket
            B_ĵ ← B_ĵ ∪ {p_i}
        end if
    end for
end for
return B
```

Thereby, we propose a general strategy to solve similarity self-join problem. It is an approximate approach based on clustering algorithm to find groups of similar elements in which the dataset is composed by high-dimensional points. There are some advantages to use clustering algorithms. The first one, a set of similar elements are represented by centroids; thus, clustering algorithm aids putting together elements that are similar to each other in the same cluster. Another one is to handle centroids instead to handle all elements from the datasets, avoiding the quadratic time in processing. We should only process centroids to assign all points from the dataset, which is less expensive, specially when using a small list of centroids. Considering that one element could be similar to several other elements, even if they are in different clusters, the bucket assignment should adopt a soft assignment strategy, in which one element is assigned to several buckets.

Basically, our solution of soft assignment, after the *K-means* clustering, can be summarized in two main steps: (i) the p points of the dataset are assigned to, at most, m nearest clusters; and (ii) the clusters contain all similar points, that can be enumerated for identifying the near-duplicate videos. Moreover, a verification step may be applied in order to

Algorithm 2 Similar pairs enumeration algorithm

Require: $\mathbf{B} = \{B_1, \ldots, B_K\}$ ▷ Buckets, $B_j \subseteq \mathbb{R}^{N \times d}$
Ensure: $\mathbf{A} \subseteq \mathbb{R}^d \times \mathbb{R}^d$ ▷ List of similar pairs

```
// Generates all pairs of points within the same bucket
A ← ∅
for l = 1 to K do
    for i = 1 to size(B_l) do
        p_i ← i-th element of B_l
        for j = i + 1 to size(B_l) do
            p_j ← j-th element of B_l
            A ← A ∪ (p_i, p_j)
        end for
    end for
end for
return A
```

remove dissimilar videos. Algorithm 1 formalizes the assignment task of our method to approximate similarity self-join with quantization of a set of high-dimensional points p, given a set of clusters identified by their centroids c. Analogously to [11], Algorithm 1 generates a compact representation of the similarity self-join result which is a set of buckets of points that satisfies the threshold of similarity-self join. Thus, for a given number of clusters, the output is linear rather than quadratic in the number of points. Moreover, Algorithm 2 can be used to enumerate all pairs of similar points if it is needed.

As can be seen in Algorithm 1, the function **isViable** controls the insertion of elements in each bucket, that could represent a similar element (for instance, a video). We have used, in this work, three different strategies to control the insertion of an element p_i in the bucket j, which is represent by the centroid c_j: (i) without any threshold; (ii) with a threshold; and (iii) with an adaptive threshold. In the first, the element p_i is included in the bucket j without any verification. For the specified threshold strategy, the element p_i is inserted in the bucket j, if $d(p_i, c_j) \leq \lambda$. Regarding the adaptive strategy, an average value of the similarities between all elements of the bucket j and the centroid c_j is used to control the insertion of the element p_i. This average value can be obtained by Equation 2,

$$d(p_i, c_j) \leq \frac{\sum_{l=1}^{\mid \hat{B}_j \mid} d(p_l, c_j)}{\mid \hat{B}_j \mid} \tag{2}$$

in which $\hat{B}_j$ represents the set of points that are already assigned to the cluster related to centroid c_j.

In terms of computational cost, the proposed Algorithm 1, runs in $O(n \times (K + K \times \log m + m))$ to solve the assignment step of the similarity self-join problem, which is the most important one. Considering that $m \leq K$, this order can be re-written as $O(n \times K \times \log m)$. It is worth to mention that this strategy identifies the groups of similar elements.

IV. Experiments

In order to evaluate the performance of our algorithm for similarity self-join, we performed two different experiments. The first one used a subset of BIGANN dataset [23], in which we studied the behavior of the proposed method by using different amount of elements, from 10,000 to 100,000 SIFT vectors. The other experiment is related to the near-duplicate video detection by using the CC_WEB_VIDEO dataset [2], [3]. This dataset consists of 13,137 videos downloaded from the Web divided into 24 categories. Differently from [2], in which the authors used the fusion of content and context information, we used only video content information to identify the near-duplicate videos. In Figure 2, we illustrate some key-frames for each category.

In order to evaluate the results on the BIGANN dataset, the ground-truth was generated by using the naive method for computing the distance between each pair of points. On the other hand, in the CC_WEB_VIDEO dataset, the ground-truth was modified: instead of representing all similar videos for a given query, we identified all pairs of similar videos through human analysis.

A. Experimental Setup

For the representation of the video, we used BossaNova mid-level representation, which is a global signature. BossaNova is a representation for content-based concept detection in images and videos, which enriches the Bag-of-Words model [16]. Here, the video is represented by a collection of SIFT descriptors [24], which are extracted from the key-frames. These SIFT descriptors are encapsulated by a BossaNova descriptor, thus each video is represented by only one descriptor. For SIFT descriptor, we obtained the code from OpenCV repository [25], and the code for BossaNova is available at http://www.npdi.dcc.ufmg.br/bossanova. BossaNova Video Descriptor was adopted to represent the video frames. We kept the BossaNova parameter values the same as in [15] and we used *max* as aggregating function.

Furthermore, in our experiments, we applied *K-means* algorithm [26] to cluster the dataset by using 30% of the elements of the dataset. In CC_WEB_VIDEO dataset, we adopted three different values for K: 16, 32, and 64. We performed experiments with each video class separately, and we did not use large values of K, since some classes are very small and a compact video representation (one high-dimensional feature per video) was adopted.

In BIGANN dataset, we adopted four different values for K: 64, 128, 256, and 512. Our soft-assignment algorithm assigns all elements to clusters. BossaNova descriptors were 3009-dimensional features for each key-frame. We performed our algorithm on subsets from BIGANN dataset with 128-dimensional SIFT points. We defined a subdataset x-BIGANN as the set of elements of BIGANN with x elements. We also applied Nested Loop (NL) algorithm in order to compare its performance to our approach. Nested Loop algorithm for similarity join computes the distance between all pairs and check if their distance is no larger than a given threshold.

In order to present a comparative analysis, we performed experiments using fixed threshold and without using threshold value to filter out pairs. As mentioned before, our method computes all centroids by *K-means* clustering and assigns each point p of dataset to the m nearest clusters. We used five

different values for m: 10%, 20%, 30%, 40%, and 50% of the number of clusters (K). The threshold value for the Nested Loop algorithm was empirically defined as 75% of the λ.

Experiments were conducted on a computer under Ubuntu 12.04 LTS OS with Intel Xeon Quad-Core 2.4 Ghz and 72 GB RAM. We evaluated the performance of our algorithm by measuring the recall and precision rates of several assessments of near-duplicate detection problem. The presented results are the average values of the five runs of our method.

B. Experimental Quantitative Results

In this subsection, we analyze the experimental results of our approximate similarity join approach. One of the main problems in dealing with Web videos is that there must exist a lot of near-duplicate videos which differ in size, format and long and short versions. The benchmark CC_WEB_VIDEO dataset brings these variations. In the clustering step of the algorithm, *K-means* is performed by a sample from all database.

Figure 3 shows the best average results of our method for similarity self-join in the BIGANN dataset experiments with K equals to 256 and m equals to 10% of the number of clusters (K). These results are expressed in terms of *F-Measure* average. Our approximate algorithm reached more than 90% of *F-Measure* in average for all datasets using fixed and adaptive threshold. This strategy avoids the similarity self-join algorithm to assign points far away to each other in the same cluster. It improves the result of the algorithm. In Table I, we present the comparative results in terms of time to process all experiments for all datasets from BIGANN. Our method outperformed, in terms of time, the Nested Loop approach missing only a couple of results. Beside that, time processing of our algorithm did not grow as Nested Loop, instead. We also present the speedup for computing the similarity self-join as a relation between the time for computing Nested Loop and the time for computing the clustering and assignment tasks.

In Figure 4 we show the best results of *F-Measure* average obtained by our method when applied to CC_WEB_VIDEO dataset with K equals to 16 and m equals to 40% of the number of clusters (K). The method without threshold

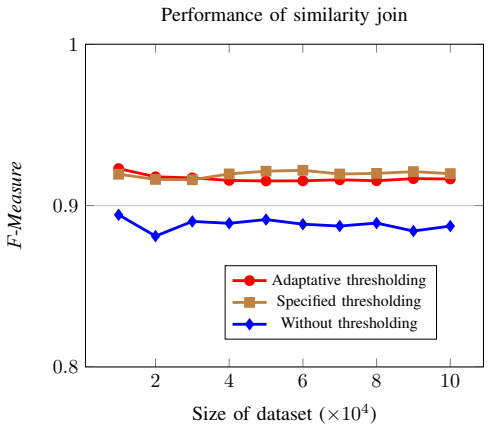

Fig. 3. *F-Measure* average results for similarity self-join in the BIGANN dataset in which $K = 256$ and $m = 10\%$ of the number of clusters (K).

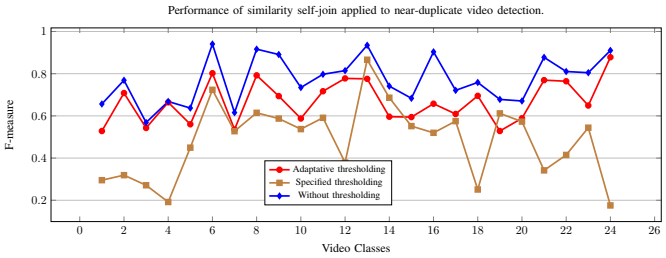

Fig. 4. *F-Measure* average results for near-duplicated video detection by using similarity self-join, in which $K = 16$ and $m = 40\%$ of the number of clusters (K).

outperformed the others in most of the video classes without losing time performance. As the experiment was performed in each video class separately, most of the videos in the same class are very similar to each other. Then, specifying a threshold (even an adaptive threshold) can deteriorate the performance. In Table II, we present the comparative results in terms of time to process all experiments for each video class from CC_WEB_VIDEO, and as expected, our method outperformed, in terms of time, the Nested Loop approach without losing performance. As we can see, our solution based on clustering can perform similarity join much faster than Nested Loop algorithm. The presented time value represents the sum of *K-means* algorithm plus similarity self-join algorithm.

V. Conclusion

In this work, we addressed the problem of near-duplicate video detection by using similarity self-join operation in order to identify groups in which the elements are similar to each other. Experimental results were used to analyze the behavior of video representation using BNVD and BIGANN dataset. The results showed that our solution can solve similarity self-join problem in approximated way with more the 90% of *F-Measure* in 100,000-BIGANN dataset reaching almost 93% of speedup in relation to nested loop strategy. We also solved near-duplicate video detection problem using approximate solution to similarity self-join, and our method outperformed the

TABLE I
TIME (SECONDS) TO PERFORM SIMILARITY SELF-JOIN FOR BIGANN DATASET USING SPECIFIED THRESHOLD STRATEGY (CLUSTERING TIME PLUS SELF-JOIN TIME), IN WHICH $K = 256$ AND $m = 10\%$ OF THE NUMBER OF CLUSTERS (K).

Size $\times 10^3$	From clustering to enumeration			NL	Speedup (NL/(C+J))
	Clustering (C)	Join (J)	Pairs		
10	1.56	3.02	0.45	109.80	23.95
20	6.90	6.13	1.77	475.64	36.48
30	13.59	9.06	3.77	1060.25	46.79
40	21.66	12.13	6.83	1908.83	56.47
50	27.65	15.12	10.29	2891.34	67.60
60	33.80	18.10	14.76	4170.56	80.34
70	42.56	21.10	19.18	5691.56	89.39
80	68.07	24.17	26.61	7424.54	80.49
90	60.51	27.04	32.19	9194.06	105.01
100	93.08	30.22	40.91	11404.03	92.49
				average	41.197

TABLE II
TIME (SECONDS) TO PERFORM SIMILARITY SELF-JOIN FOR CC_WEB_VIDEO DATASET WITHOUT THRESHOLD STRATEGY (CLUSTERING TIME PLUS SIMILARITY SELF-JOIN TIME), IN WHICH $K = 16$ AND $m = 40\%$ OF THE NUMBER OF CLUSTERS (K).

Video class	From clustering to enumeration			NL	Speedup
	Clustering (C)	Join (J)	Pairs		(NL/(C+J))
1	0.121	0.326	0.004	13.240	29.62
2	0.127	0.239	0.002	7.360	20.11
3	0.049	0.180	0.001	4.130	18.03
4	0.040	0.142	0.001	2.570	14.12
5	0.054	0.164	0.001	3.410	15.64
6	0.251	0.317	0.003	13.038	22.95
7	0.049	0.151	0.001	2.894	14.47
8	0.128	0.221	0.002	6.196	17.75
9	0.042	0.098	0.001	1.206	8.61
10	0.068	0.123	0.001	1.917	10.04
11	0.034	0.156	0.001	3.087	16.25
12	0.192	0.360	0.004	16.993	30.78
13	0.078	0.172	0.001	3.732	14.93
14	0.026	0.045	0.001	0.246	3.46
15	0.422	0.725	0.017	68.796	59.98
16	0.019	0.086	0.001	0.917	8.73
17	0.115	0.267	0.002	9.155	23.97
18	0.077	0.202	0.002	5.190	18.60
19	0.075	0.234	0.002	6.957	22.51
20	0.023	0.081	0.001	0.821	7.89
21	0.079	0.184	0.001	4.383	16.67
22	0.067	0.174	0.001	3.882	16.11
23	0.315	0.540	0.011	38.344	44.85
24	0.047	0.119	0.001	1.794	10.81
average	0.104	0.221	0.003	9.177	28.22

nested loop approach, in terms of time, without a significant loss of performance, in terms of *F-Measure*.

As further work, we consider to explore this kind of strategy to cope with very large datasets, mainly by studying the behavior of other clustering methods and other way to define the insertion in the buckets. Furthermore, we also intend to study the amount of redundant pairs that may be generated by our enumeration algorithm.

ACKNOWLEDGMENTS

The authors are grateful to FAPEMIG/INRIA/MOTIF, CNPq, CAPES and STIC-AmSud for the partial financial support of this work.

REFERENCES

[1] H. Wang, F. Zhu, B. Xiao, L. Wang, and Y.-G. Jiang, "GPU-based MapReduce for large-scale near-duplicate video retrieval," *Multimedia Tools and Applications*, vol. 74, no. 23, pp. 10 515–10 534, 2015.

[2] X. Wu, C.-W. Ngo, A. G. Hauptmann, and H.-K. Tan, "Real-Time Near-Duplicate Elimination for Web Video Search With Content and Context," *IEEE Transactions on Multimedia*, vol. 11, no. 2, pp. 196–207, 2009.

[3] X. Wu, A. G. Hauptmann, and C.-W. Ngo, "Practical elimination of near-duplicates from web video search," in *Proceedings of the international conference on Multimedia - MM '07*. New York, NY, USA: ACM Press, 2007, pp. 218–227.

[4] C. L. Chou, H. T. Chen, and S. Y. Lee, "Pattern-Based Near-Duplicate Video Retrieval and Localization on Web-Scale Videos," *IEEE Transactions on Multimedia*, vol. 17, no. 3, pp. 382–395, 2015.

[5] L. Shang, L. Yang, F. Wang, K. P. Chan, and X. S. Hua, "Real-time large scale near-duplicate web video retrieval," in *Proceedings of the international conference on Multimedia - MM '10*. New York, NY, USA: ACM Press, 2010, pp. 531–540.

[6] J. Song, Y. Yang, Z. Huang, H. T. Shen, and R. Hong, "Multiple feature hashing for real-time large scale near-duplicate video retrieval," in *Proceedings of the International Conference on Multimedia - MM '11*. New York, NY, USA: ACM Press, 2011, pp. 423–432.

[7] Y. Jiang, G. Li, J. Feng, and W. Li, "String Similarity Joins: An Experimental Evaluation," in *Proceedings of the VLDB Endowment*, 2014, pp. 625–636.

[8] 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, pp. 1–41, 2011.

[9] Y. Kim, K. Shim, M.-S. Kim, and J. Sup Lee, "DBCURE-MR: An efficient density-based clustering algorithm for large data using MapReduce," *Information Systems*, vol. 42, pp. 15–35, 2014.

[10] D. Chen, L. Zheng, M. Zhou, and S. Yu, "Efficient Similarity Join for Time Sequences Using Locality Sensitive Hash and Mapreduce," in *2013 International Conference on Cloud Computing and Big Data*. IEEE, 2013, pp. 529–533.

[11] B. Bryan, F. Eberhardt, and C. Faloutsos, "Compact Similarity Joins," in *IEEE 24th International Conference on Data Engineering*, 2008, pp. 346–355.

[12] A. Arasu, V. Ganti, and R. Kaushik, "Efficient exact set-similarity joins," in *Proceedings of the 32Nd International Conference on Very Large Data Bases*. Seoul, Korea: VLDB Endowment, 2006, pp. 918–929.

[13] C. Rong, W. Lu, X. Wang, X. Du, Y. Chen, and A. K. Tung, "Efficient and Scalable Processing of String Similarity Join," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 10, pp. 2217–2230, 2013.

[14] M. Wang, T. Nie, D. Shen, Y. Kou, and G. Yu, "Intelligent Similarity Joins for Big Data Integration," in *10th Web Information System and Application Conference*. IEEE, 2013, pp. 383–388.

[15] C. Caetano, S. Avila, S. Guimarães, and A. A. Araujo, "Pornography detection using BossaNova video descriptor," in *Signal Processing Conference (EUSIPCO), 2014 Proceedings of the 22nd European*, Lisbon, 2014, pp. 1681–1685.

[16] S. Avila, N. Thome, M. Cord, E. Valle, and A. de A. Araújo, "Pooling in image representation: The visual codeword point of view," *Computer Vision and Image Understanding*, vol. 117, no. 5, pp. 453–465, 2013.

[17] S. Fries, B. Boden, G. Stepien, and T. Seidl, "PHiDJ: Parallel similarity self-join for high-dimensional vector data with MapReduce," in *IEEE 30th International Conference on Data Engineering*, 2014, pp. 796–807.

[18] D. Deng, G. Li, S. Hao, J. Wang, and J. Feng, "MassJoin: A mapreduce-based method for scalable string similarity joins," in *IEEE 30th International Conference on Data Engineering*, 2014, pp. 340–351.

[19] J. Feng, J. Wang, and G. Li, "Trie-join: a trie-based method for efficient string similarity joins," *The VLDB Journal*, vol. 21, no. 4, pp. 437–461, 2011.

[20] W.-L. Zhao, H. Jégou, and G. Gravier, "Sim-min-hash," in *Proceedings of the International Conference on Multimedia - MM '13*. New York, New York, USA: ACM Press, 2013, pp. 577–580.

[21] V. Satuluri and S. Parthasarathy, "Bayesian Locality Sensitive Hashing for Fast Similarity Search," in *Proc. VLDB Endow.*, vol. 5, 2012, p. 13.

[22] J. Gao, H. V. Jagadish, W. Lu, and B. C. Ooi, "DSH: Data Sensitive Hashing for High-dimensional K-nnsearch," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data - SIGMOD '14*. New York, USA: ACM Press, 2014, pp. 1127–1138.

[23] H. Jégou, M. Douze, and C. Schmid, "Product Quantization for Nearest Neighbor Search," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 1, pp. 117–128, 2011.

[24] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[25] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, vol. 25, no. 11, pp. 122–125, 2000.

[26] M. Douze and H. Jégou, "The Yael library," in *ACM Multimedia*, 2014.