Li *et al. EURASIP Journal on Image and Video Processing* (2019) 2019:51
https://doi.org/10.1186/s13640-019-0442-7

EURASIP Journal on Image and Video Processing

RESEARCH Open Access

Fast distributed video deduplication via locality-sensitive hashing with similarity ranking

Yeguang Li[1,2], Liang Hu[1], Ke Xia[3] and Jie Luo[3*]

Abstract

The exponentially growing amount of video data being produced has led to tremendous challenges for video deduplication technology. Nowadays, many different deduplication approaches are being rapidly developed, but they are generally slow and their identification processes are somewhat inaccurate. Till now, there is rare work that studies the generic hash-based distributed framework and the efficient similarity ranking strategy for video deduplication. This paper proposes a flexible and fast distributed video deduplication framework based on hash codes. It is able to support the hash table indexing using any existing hashing algorithm in a distributed environment and can efficiently rank the candidate videos by exploring the similarities among the key frames over multiple tables using MapReduce strategy. Our experiments with a popular large-scale dataset demonstrate that the proposed framework can achieve satisfactory video deduplication performance.

Keywords: Video deduplication, Distributed computing, Locality sensitive hashing, Hash table indexing, Similarity ranking

1 Introduction

Due to the increasing popularity of mobile devices and social networks, huge numbers of videos are being created and shared online. This explosive growth in the amount of video data being produced has made storing and rapidly searching it all very challenging. In practice, many videos are duplicates, or near-duplicates, so detecting these copies has become a very important technique for reducing the storage and computation required.

In recent years, many content-based duplicate detection techniques have been developed that aim to identify such copies automatically in massive datasets. For example, a million-video-scale near-duplicate video retrieval system [1] has been developed that quantizes the key frames' features into visual words and uses an inverted file index to implement rapid search. In contrast to prior research, Song et al. [2] presented a near-duplicate video retrieval method based on compact hash codes learnt from multiple visual features, a promising solution that enables fast signature generation based on binary codes from multiple views.

Hashing-based approximate nearest-neighbor search has attracted much attention in the literature, owing to its high search performance and low computational requirements [3–6]. Locality-sensitive hashing (LSH) is the most fundamental concept in such hashing research [7], which has led many studies to focus on finding ways to generate compact hash codes by exploiting different techniques, including (semi-) supervised learning [8–10], non-linear mapping [5, 8, 11–15], discrete optimization [12, 16–18], multiple features [2, 19], and bit selection [20, 21].

Most of the existing hashing methods have been proposed to handle the popular vectorial data like the images. They can also be directly applied to indexing the gigantic video data by treating frames as images [11, 22–25]. Song et al. [2] proposed a multiple feature-based hashing to capture different aspects of visual content. Cao et al. [23] effectively selected a number of informative features to characterize the video content under a submodular hashing framework. Xia et al. and Wang et al. [26, 27]

*Correspondence: luojie@nlsde.buaa.edu.cn
[3]State Key Lab of Software Development Environment, Beihang University, Beijing, China
Full list of author information is available at the end of the article

further employed the subspace representation to generate framewise hash code by considering local structure of the consecutive frames. However, these video hashing solutions only take the video frames as the still images and generate their binary codes independently. In practice, it is well known that videos are quite different from images with more complex semantic and temporal information. The video deduplication should take the temporal order of the frames into the consideration.

Despite this progress of the highly developed hashing techniques, little attention has been paid to methods of building efficient indexes with hash codes or generating good ranking lists by aggregating results from multiple indexes. There are a few studies that attempt to address the image indexing by learning a number of the complementary hash tables [19, 28, 29]. However, these techniques heavily rely on the specific and usually expensive learning algorithms, which can hardly be compatible to the generic scenarios and existing hashing algorithms.

In this paper, we address this problem by proposing a generic, yet fast video deduplication framework based on hash codes. This framework supports hash table indexing and searching based on any existing binary hashing algorithm, and it can adaptively combine ranking results from multiple tables by considering key frame similarities. To the best of our knowledge, this is the first attempt to study a general hashing-based video deduplication framework that can support large-scale video databases.

To handle the large-scale duplication problem, the distributed computing is popular and successful technique in the literature. Kumar et al. [30] proposed a technique whereby a chunking algorithm divides the data stream into fixed-size chunks, from which hash values are generated via the MD5 algorithm and then used by a MapReduce (MR) model to identify duplicates. Moise et al. [31] proposed using MR for efficient index creation and search, enabling billions of descriptors to be indexed and large batches of queries to be processed. Following the prior research, in this paper, we further enhance our hashing-based video deduplication method with a distributed framework, which simultaneously exploits both the computing power of the distributed nodes and the nature distributed storage of the gigantic video data nowadays.

Note that the whole paper extends upon a previous conference publication [32] with additional exploration and experiments on the general distributed computing framework for the hash-based video deduplication. The rest of this paper is organized as follows. Section 2 introduces the proposed hashing-based video deduplication framework. Section 3 elaborates our approach to fine-grained ranking over multiple table indexes, which attempts to capture the videos' similarities. Section 4 describes how we use MapReduce to process the video data both online and offline. Section 5 presents the results of experiments on a popular benchmark, demonstrating the proposed method's effectiveness. Finally, Section 6 concludes the paper.

2 Methods - hash-based video deduplication framework

In this section, we first outline our framework and its main components, then introduce the hashing-based video indexing process used by the framework.

2.1 Video deduplication framework

Figure 1 gives an overview of our framework for large-scale video deduplication. Given a query video, this can efficiently find matches between that video and those in the database, rapidly identifying duplicate videos. The framework consists of four main components: video hashing, index construction, video archiving, and video deduplication. These components perform the following functions.

Video hashing: In this step, we process the query video by first extracting key frames, then generating visual features (via the Color and Edge Directivity Descriptor (CEDD) approach in this paper) for each one. Based on these features, we represent each key frame by a set of binary hash codes from different hashing functions. Many different hash functions could be used here, such as projection-based [16, 33] or prototype-based [34] functions.

Index construction: Using multiple hash tables has been found to be very helpful for achieving high recall performance for big data search [35, 36]. In this step, we therefore build multiple hash tables based on the binary codes obtained above. Again, there are many possible strategies for this, including multi-index hashing [37] and complementary hash tables [19, 28].

Video archiving: Using the above two steps, all the videos in the database are represented as binary codes and imported into multiple hash tables. Figure 2 shows the structure of these indexes, where each unique hash code corresponds to a bucket containing similar key frames. This step is carried out offline.

Video deduplication: The given query video is first hashed to generate a set of binary codes for each key frame. Then, for each hash code, we check all the buckets in the corresponding hash table within a small Hamming distance of it, considering the videos containing the key frames in these buckets as candidate results. We then rank them by similarity (Section 3), enabling duplicate videos to be easily detected.

2.2 Hash table indexing

One of the most important parts of the above framework is the hash table indexing step, as it guarantees low memory consumption and satisfactory deduplication per-

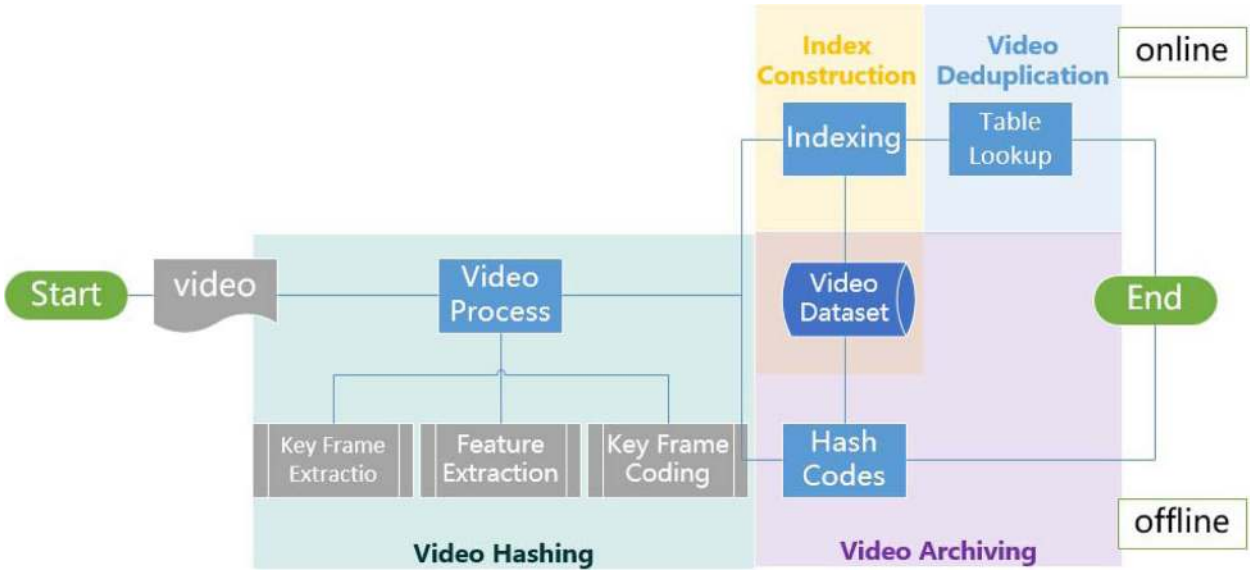


Fig. 1 Proposed hash-based framework for fast video deduplication

formance. To achieve the desired performance, we combine several efficient techniques, including multiple-table indexing, multi-probe search, and Hamming distance-based ranking.

Suppose that, for each key frame $\mathbf{x}$, we generate a set of B binary hash codes $\mathbf{y} = [h_1(\mathbf{x}), \ldots, h_B(\mathbf{x})] \in \{-1, 1\}^M$. Many different hash functions h_b, $b = 1, \ldots, B$ can be used, of which the simplest is random projection. Then, we build L hash tables by evenly partitioning the code $\mathbf{y}$ into L subcodes of length $\frac{M}{L}$, which is usually less than 32 in practice. These subcodes can then be used to build a set of hash tables $\{\mathcal{T}_l, l = 1, \ldots, L\}$, each based on hashes of length $\frac{M}{L}$. In these hash tables, each bucket contains key frames from videos in the database.

Figure 2 shows an example hash table based on 4-bit codes, where a total of 2^4 buckets each store the key frames extracted from different videos that share the same hash code. Using these tables, we perform separate hash

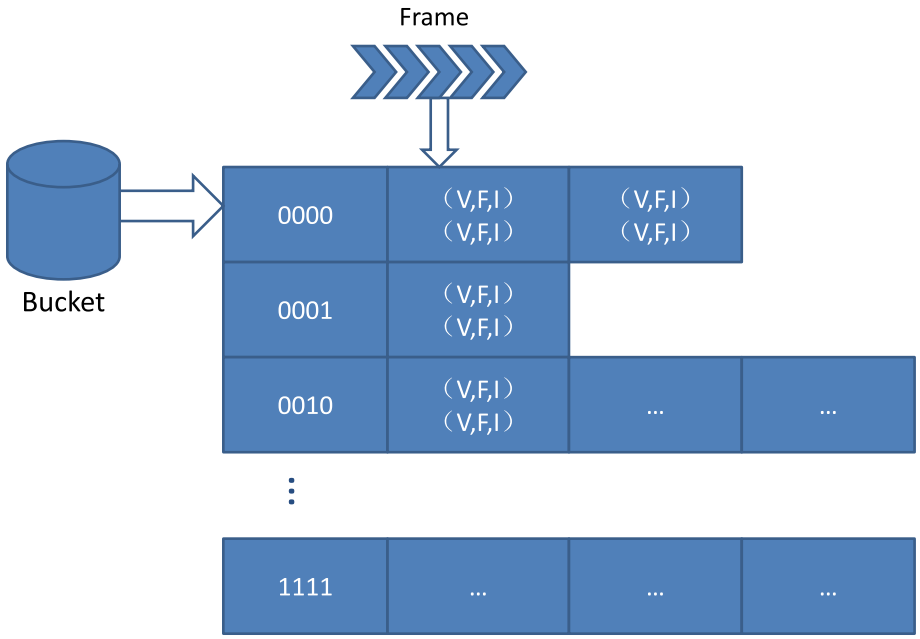


Fig. 2 Hash table structure used for the video index

table lookups for each key frame in the given query video to efficiently find similar frames. Based on these search results, we can then determine whether the query video is a duplicate of one in the database. In the next section, we discuss in detail how table lookups are performed and the final results are generated.

3 Similarity ranking over multiple tables

The multiple-hash-table lookup process returns several different result sets, so one critical problem is how to combine these to form the final ranking list. This section describes our similarity ranking strategy.

3.1 Frame similarity

During the online search phase, we first divide each key frame's hash code into L equal parts, then look up each subcode in its corresponding table. The most common table lookup strategy is to search all buckets within a small radius. Therefore, given an allowed lookup radius R, the maximum Hamming distance for each table is $\frac{R}{L}$.

To compute the similarity of two videos, we should first capture the similarity relationships among their key frames. For the ith key frame $\mathbf{x}_{qi}$ of the query video and the$j$th key frame $\mathbf{x}_{dj}$ of the dth video in the database, we combine the Hamming distance-based similarities for each hash table to derive the overall similarity of the key frame pair in the natural way:

$$s_{ij} = 2 * R - \sum_{l=1}^{L} \left\{ \alpha r_l + (1-\alpha) \left(\frac{R}{L} + 1 \right) \right\} \tag{1}$$

where r_l is the Hamming distance between the l-th subcodes of the query and database frames and α is a weighting parameter that controls the contribution of the Hamming distance. Essentially, when $r_l \leq \frac{R}{L}$, the videos are highly likely to be duplicates, so we set $\alpha = 1$; otherwise, they are unlikely to be duplicates, so we set $\alpha = 0$.

In summary, the above similarity definition is based on the fact that when the subcode distances are smaller and more key frames match, the videos are more likely to be duplicates.

3.2 Video similarity

The above similarity definition ignores the videos' temporal information, but prior research has demonstrated that including such information can improve performance [38]. Therefore, we also consider the temporal consistency between the matched frames. However, considering high-order temporal sequences is quite complex and time-consuming, so we simplify the problem by focusing only on the temporal orders of pairs of frames, instead of the full list.

Specifically, we define the order preservation ratio (OPR) as follows:

$$o_{ij}^{d} = \frac{|I_{dk_i'} - I_{dk_j'}|}{|I_{qk_i} - I_{qk_j}|} \tag{2}$$

Here, we consider two pairs of matched frames, namely, the k_ith and k_jth frames of the query video and the corresponding k_ith and k_jth frames of the dth database video, denoting the order of the k_i'th frame in the dth video by $I_{dk_i'}$ and defining the other variables likewise. This captures the idea that the key frames of two duplicate videos will be consistently ordered in the fact that OPRs for matched frame pairs will then remain constant. We can therefore use this to filter out false positive video matches.

Based on this intuition, we refine the similarity metric between the query and dth database videos by considering the temporal order of all possible pairs of two matched frames and simply summing the similarities of pairs with the same OPR. This means that if more frames match in consistent order, they will contribute more to the similarity. If we have two sequences of matched frames, $\{I_{qk_1}, \ldots, I_{qk_m}\}$ from the query video and $\left\{I_{dk_1'}, \ldots, I_{dk_m'}\right\}$ from the database video, we calculate the similarity matrix $\mathbf{G}^d = \left(g_{ij}^d\right)$, $1 \leq i,j \leq m$ as follows. Each entry g_{ij}^d sums the similarities of the individual frames in a given matched pair:

$$g_{ij}^{d} = s_{k_i k_i'} + s_{k_j k_j'}. \tag{3}$$

Therefore, after computing $\mathbf{G}$ and all the o_{ij}^d for the matched video sequence, we sum all the h_{ij}^d with the same o_{ij}^d, forming the histogram $\mathbf{h}^d = \left(h_i^d\right)$:

$$h_i^d = \sum_{o_{ij}^d = v_i^d} g_{ij}^d, \tag{4}$$

where v_i^d is an OPR value.

Figure 3 shows two cases, one where multiple matched-frame pairs share the same OPR and another where they do not. Here, we can see that sequences where more frames match in a consistent order will have higher similarities due to the summation process, helping us to distinguish true duplicates from false positives with less-consistent frame matches.

Based on the histogram, we obtain the final similarity between the query and database videos by taking the histogram's maximum value, which reveals the dominant matching order. To eliminate the effect of video length, we normalize the similarity as follows:

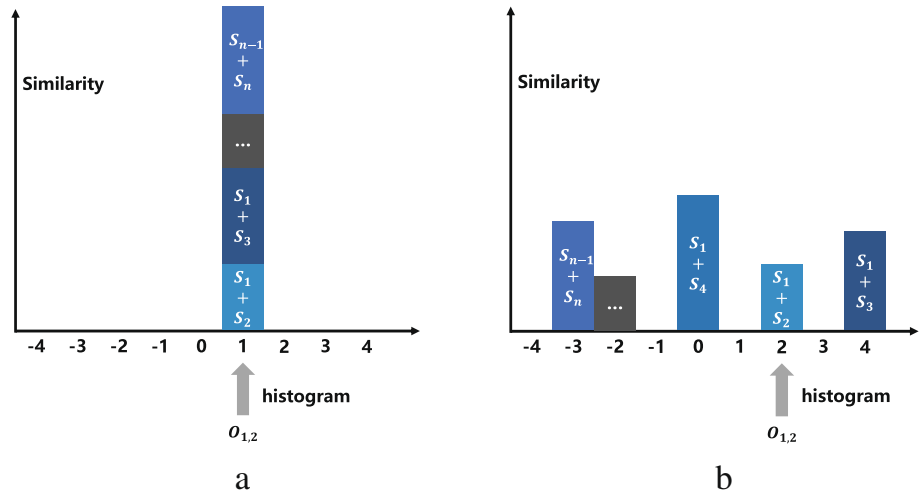


Fig. 3 Similarity histograms, showing similarity distributions based on the OPR. **a** same OPR. **b** different OPRs

$$S_d = \frac{1}{m} \max_i h_i^d \tag{5}$$

where m is the number of matched key frames for the query and database videos. Based on this similarity metric, we can easily rank all candidates in descending order. Since the candidate set is quite small relative to the size of the database, by efficiently computing the Hamming distance and OPR, we can generate this similarity ranking quite quickly.

4 Distributed deduplication

The distributed framework proposed in this paper is based on the MR model, which assigns tasks equally to each Hadoop DataNode. In this section, we will elaborate how MR is used for distributed video processing, during both the offline and online stages.

4.1 Offline video data processing

First, all the videos in the database are processed, and these tasks are assigned to an average of M DataNodes, one per video. When the initial preprocessing step is complete on each DataNode, hash codes are generated for each key frame and L DataNodes are allocated to build hash tables. Next, the key frame hash codes generated for the M videos are looked up in the L hash tables by MR, and the matching results are obtained. Finally, the video storage process is completed based on the matching results.

The video preprocessing and hash table creation processes are described in detail above, so now we discuss how MR is used to perform the matching operations. Early in the map processing phase, the input data is split into groups by the InputSplit method and parsed into intermediate key/value pairs, which are then used as input to the reduce method in order to obtain the final results.

During the map phase, we generate key/value pairs with a subcode as the key and the ID number of the corresponding key frame (VF) as the value (<subcode, VF> in Fig. 4). The shuffle step then regroups key frames with the same hash codes to generate an intermediate sequence of key/value pairs (<subcode, [VF]>). Then, the reduce phase looks up each key in the appropriate hash table, yielding pairs of hash table entries (buckets) and key frame groups (<bucket, [VF]>). Finally, the ID numbers [VF] of key frame groups with same hash code are stored in the corresponding hash table.

4.2 Online video data processing

First, we preprocess the input video to generate key frame hash codes, then we look up these hash codes in the hash tables and obtain the matching results using MR (Fig. 5). Here, the keys and values are the hash table TL and the corresponding hash code for the input video.

As shown in Fig. 6, the input data is first parsed into a series of key/value pairs <K1, V1>, where K1 represents the hash table TL and V1 represents the key frame's subcode. The map phase creates a sequence of L key/value pairs based on K1, namely <K2, List (V2)>, which give the key frame IDs V2 corresponding to the subcode K2. The reduce phase traverses the list of key frames V2 corresponding to subcode K2, as follows:

for (V2 = first, V2 != NULL, V2 = V2.next)
    if V2 = lookup(K2), emit<K2,V2>;

Finally, we obtain key/value pairs <K2, V2> of hash table entries (buckets) and key frames that give, for each subcode K2, the similar key frames stored in the same bucket.

Figure 6 demonstrates the case of hash table lookup for MapReduce-based. Define the hash table T_L as the key, and corresponding queried hash code is the value . First,

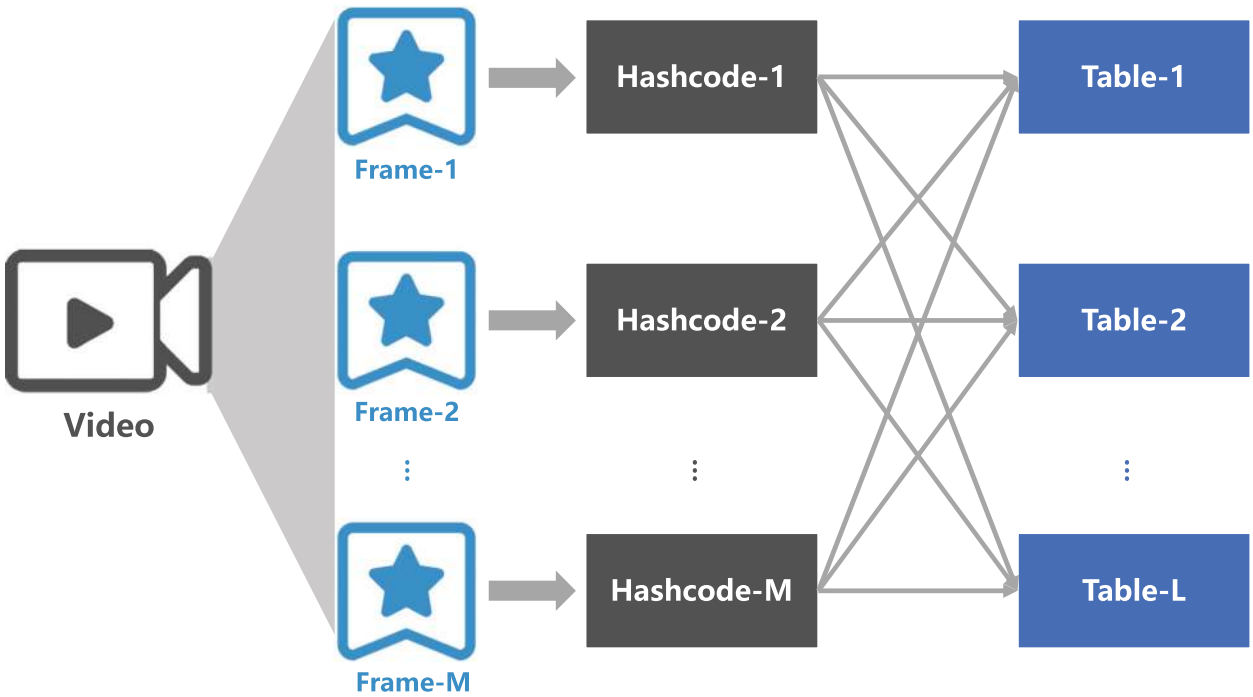


Fig. 4 Data processing with MapReduce

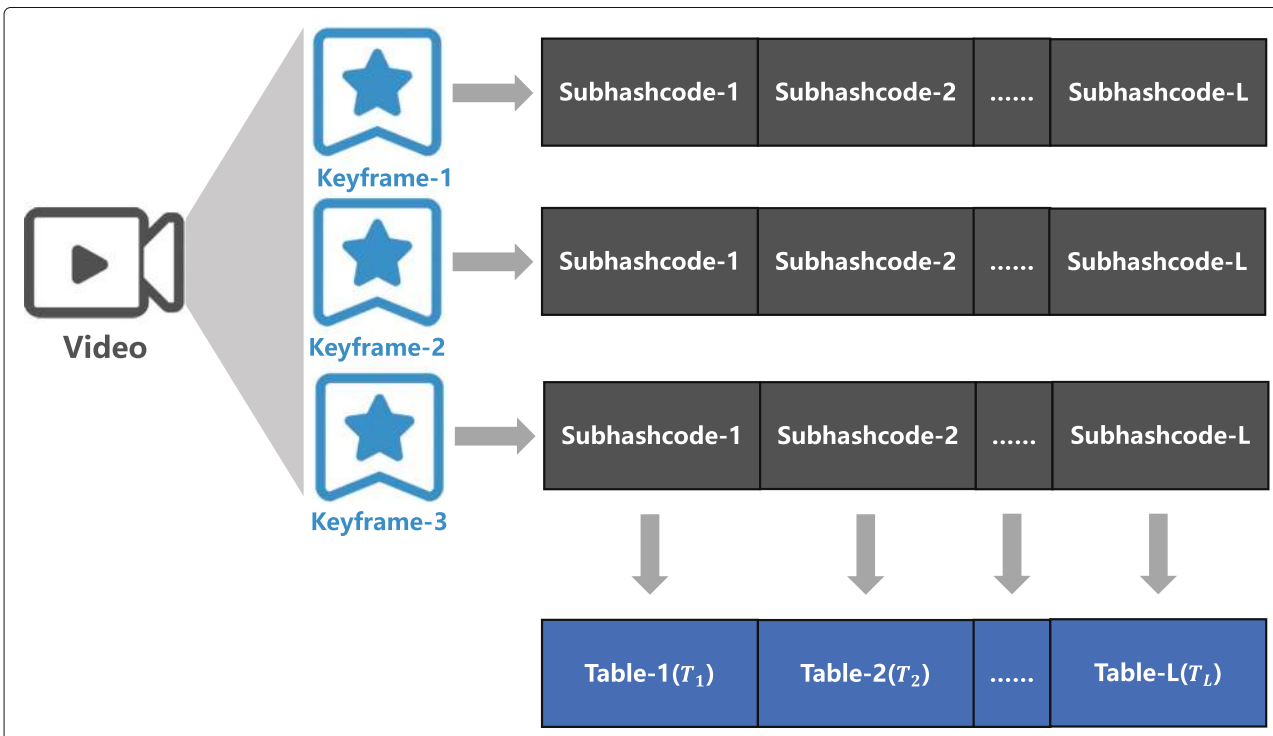


Fig. 5 Overview of the online video processing

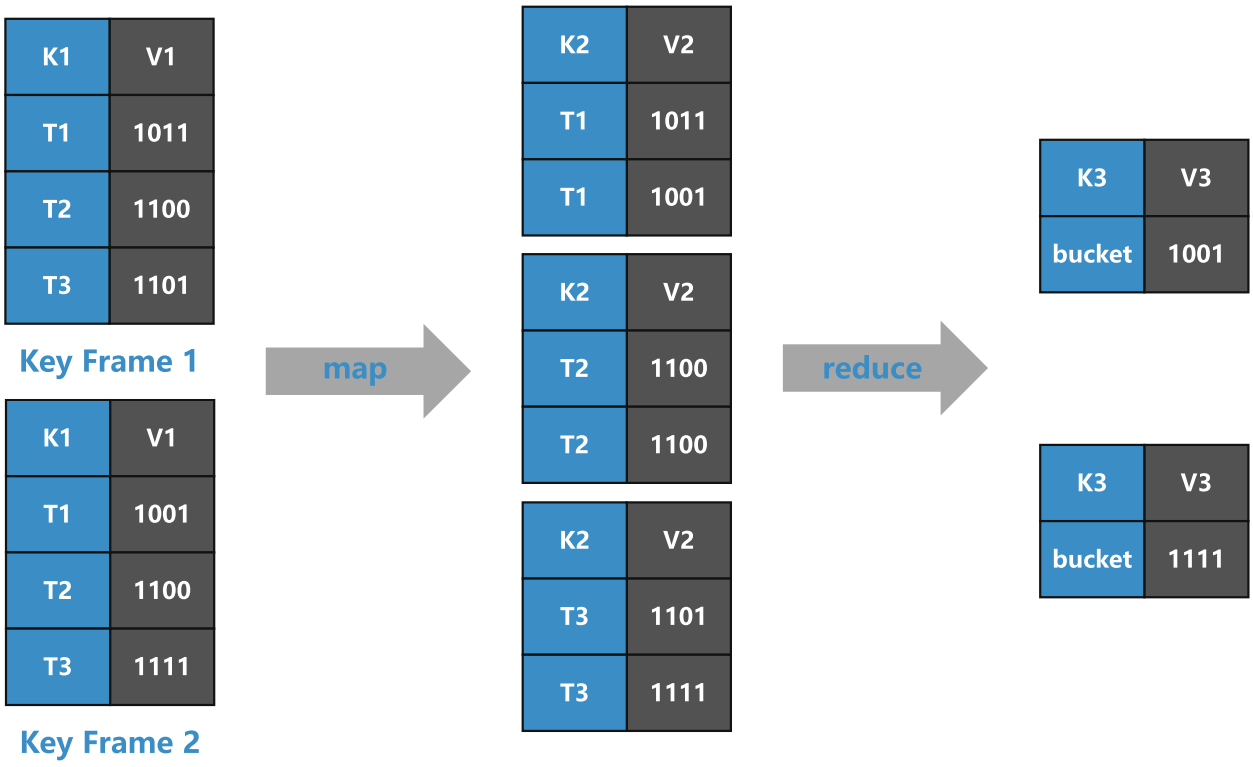


Fig. 6 MapReduce process for online video processing

the input data is parsed into a series of key-value pairs :<K1, V1>, K1 represents the hash table T_L, V1 represents the sub hash code of key frame. Through the map phase, a sequence of key-value pairs < K2, List (V2) > are grouped by K1. In the reduce phase, every V2 in the corresponding K2 will be traversed. For (V2=first, V2!=NULL, V2=V2.next) if V2=lookup(K2), emit<K2,V2>. Finally, the key-value pairs <K3,V3> of hash index entries (buckets and sub hash codes) are obtained.Thus, for each V3, there are the similar key frames stored in K3.

5 Results and discussions

In this section, we evaluate the proposed framework on a large-scale video deduplication task.

Datasets and protocols: Here, we adopted the widely used UQ_VIDEO [2], which is a combined video dataset created from the CC_WEB_VIDEO [39] by adding videos downloaded from YouTube. The YouTube videos were selected based on the most popular queries from the Google Zeitgeist Archives from 2004–2009. The UQ_VIDEO dataset contains a total of 169,952 videos, making it the largest web video dataset designed for experiments. In addition, it provides 3,305,525 key frames extracted from these videos. For testing, CC_WEB_VIDEO contains 24 manually defined near-duplicate web videos for use as queries. For each query video, there are several ground truth videos that are identical or nearly identical, but different in terms of features such as the file format, encoding parameters, editing operations, or length.

With regard to performance metrics, we employed the common precision and recall metrics for hash table lookup. Essentially, we retrieved all key frames that fell into the buckets of any table within a given Hamming radius of the query hash code. In most experiments, we used the popular LSH algorithm to generate the hash codes. However, we also investigated the effect of using different hashing algorithms, including iterative quantization (ITQ) [16]. We tried building several different numbers of hash tables from the hash codes, each using codes of different lengths.

Code length: First, we studied the effect of changing each table's code length. In this experiment, we built four tables, with code lengths of 16, 20, and 24, using 64, 80, and 96 LSH functions. Figure 7 shows their precision-recall performance for lookup radii of $D = 0$, $l = 1$, $\ldots, 4$. These precision-recall curves show the overall video duplicate detection performance, with larger areas under the curve indicating better performance. Here, we can see that 96-bit hash codes achieved the best performance in both respects, which is consistent with the fact that the optimal code length value should close to $\log_2 n$,

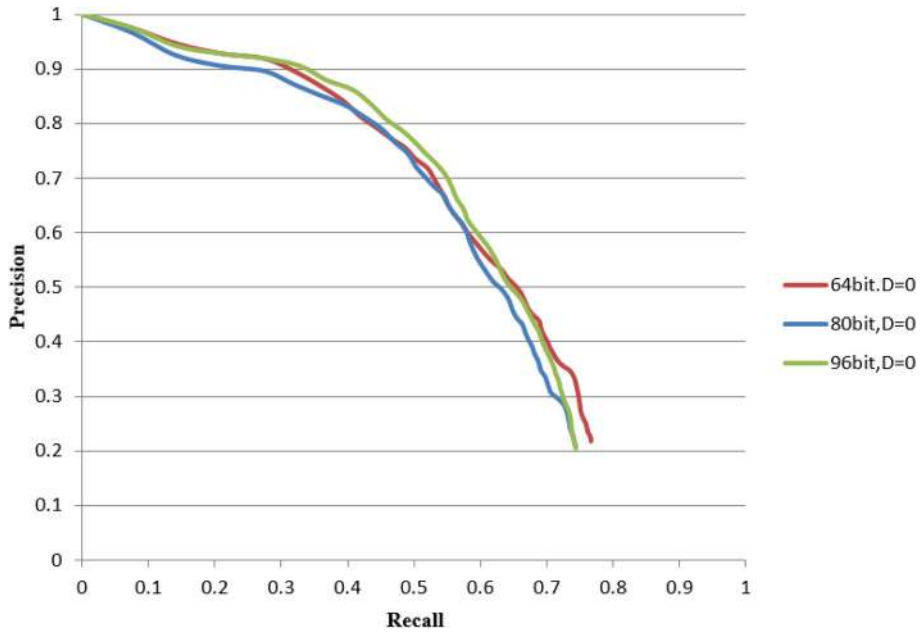


Fig. 7 Performance using hash codes of different lengths

where n is the number of frames [37, 40] (3,305,525 in this case).

Even though we obtained the best performance with 96-bit functions, the results were also satisfactory with 64 and 80 bits. However, using fewer hash bits for each table means more frames fall into each bucket, leading to increased computation costs for the similarity ranking process, due to the larger number of candidate-matched frame pairs. Therefore, in practice, the ideal code length should be a balance between high performance and fast execution. Table 1 lists the computation times in all three cases, showing that, as the code length increased, the time required for similarity ranking decreased. In addition, increasing the search radius increased the time taken significantly. This is mainly because both short codes and large radii increase the collision probability, leading to more candidates for similarity ranking. After balancing the precision-recall performance with the computational cost, we chose to use 80-bit hash functions for all other experiments.

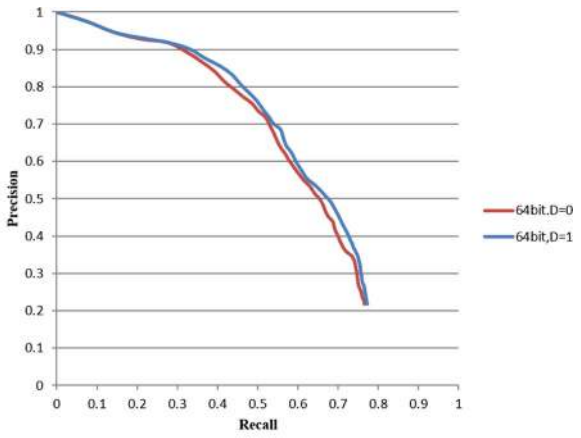
Table 1 Similarity rankings and encoding times using 64, 80, and 96 hash bits

Code length	Similarity ranking (s)		Hash coding (ms)
	$D = 0$	$D = 0$	
64	3.75	11.5	5.34
80	1.77	5.81	7.87
96	1.56	4.33	9.83

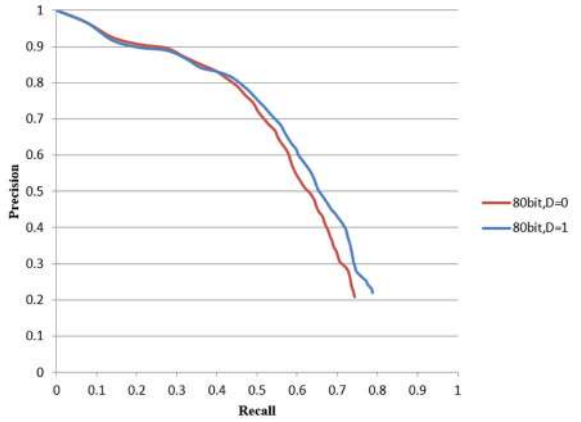
Lookup radius: In addition to the hash length, we also considered the effect of changing the lookup radius. Figure 8 shows the results of using different lookup radii (D) with different hash lengths. Here, we can easily see that increasing the search range improves the overall performance, especially the recall performance. This is because more candidates participate in the similarity ranking process, enabling our ranking method to distinguish duplicates more easily from this larger candidate set.

Hashing algorithms: In the literature, any well-designed hashing algorithms have been proposed and shown to be more powerful than the basic LSH approach in many applications. We therefore compared the LSH-based video deduplication results above with those of state-of-the-art hashing algorithms. Here, we examined the most successful of them, namely, iterative quantization hashing (ITQ), to demonstrate the effect of different hashing algorithms on our task.

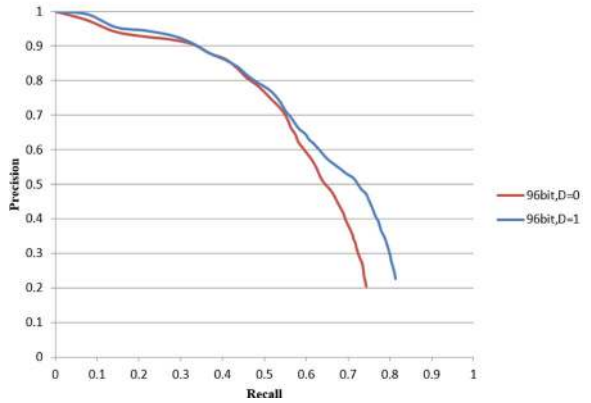
Figure 9 shows the relative performance of LSH and ITQ. We can clearly see that LSH yields better performance than ITQ, indicating that, when building multiple tables, LSH may actually be better even than well-designed hashing algorithms like ITQ. We believe this is mainly because these methods were not originally designed for multiple tables, and thus ignore table complementarity [19, 28]. The computational costs of using LSH and ITQ also emphasize this point, as LSH only took 1.77 s, while ITQ required 12.81 s due to a lack of discriminative power when building multiple tables.



a



b



c

Fig. 8 Precision-recall performance for different lookup radii, with **a** 64, **b** 80, and **c** 96 LSH bits

Similarity ranking: Next, we evaluated the proposed similarity ranking method. We compared it with a naive baseline approach that scores the candidate videos based on a basic voting strategy, without considering the temporal order or Hamming distance. Figure 10 shows the results, demonstrating that the proposed method significantly outperformed the native solution, which has been widely used in hash-based applications. Since our method can be applied to arbitrary data sequences, it could be beneficial in many similar applications in the future.

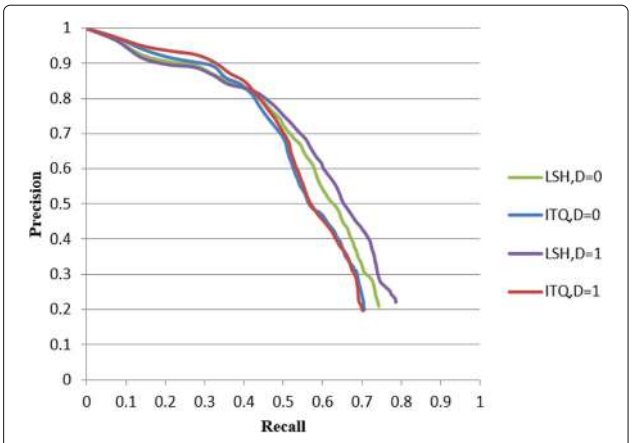


Fig. 9 Performance comparison for LSH and ITQ hashing

Computational cost: Compared with the processing time required by a single machine, using MR for large-scale video data processing is more efficient. In this study, we created an MR test environment consisting of four physical machines, one NameNode and three DataNodes, each configured as shown in Table 2.

The MR model must copy the data from disk to the Hadoop Distributed File System (HDFS) when dealing with video data, which delays processing by the time required to copy the data across the network. We therefore carried out the preprocessing steps on disk instead of copying the data to the HDFS, which greatly reduced the time required to copy the data.

Figure 11 shows that MR had a clear processing speed advantage when dealing with more than 100 million

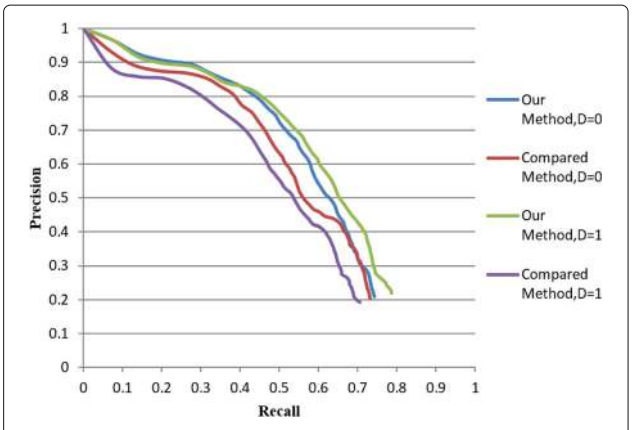


Fig. 10 Performance comparison for the proposed and baseline similarity ranking methods

Table 2 Node configuration used for MR

Cpu	2*Intel Xeon 4 Core E5520 2.26 GHz
Memory	8*2GB DDR3 1066 MHz
Hard disk	2*146GB SCSI
Adapter	1000MB Ethernet
OS	Ubuntu 12.02 64 bits
Java version	1.6.0
Hadoop version	1.2.1

of video frames, and that this steadily increased with the amount of video. For example, with 900 million of video frames, MR was 44.2% faster than using a single machine.

6 Conclusions

To achieve the fast deduplication of a large-scale video dataset, this paper proposed a distributed framework based on locality-sensitive hashing, which is generic and powerful to use any existing hashing algorithm to build multiple hash table indexes. Based on the efficient indexes, we then developed an efficient similarity ranking method that combines the search results from multiple tables by considering both the Hamming distances between key frames and the frames in temporal order. By further introducing the distributed computing strategy based on the MapReduce, the efficiency of hash-based deduplication is further improved at both offline indexing and online search stages. We conducted several experiments on large-scale video datasets to evaluate the different aspects of our method, and the results indicate that the proposed method is robust and efficient for large-scale video deduplication.

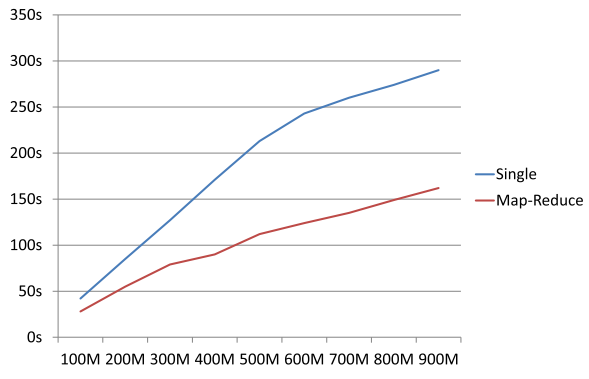


Fig. 11 Comparison between single machine and MR processing performance

Acknowledgements
Not applicable.

Funding
This work was supported by National Natural Science Foundation of China (61690202, 61701190 and 61872021), National Key R&D Plan of China (2017YFA0604500), National Sci-Tech Support Plan of China (2014BAH02F00), Youth Science Foundation of Jilin Province of China (20160520011JH and 20180520021JH), Youth Sci-Tech Innovation Leader and Team Project of Jilin Province of China (20170519017JH), Key Technology Innovation Cooperation Project of Government and University for the whole Industry Demonstration (SXGJSF2017-4), and Key Scientific and Technological R&D Plan of Jilin Province of China (20180201103GX).

Availability of data and materials
Please contact author for data requests. The dataset used in the current study are available from http://staff.itee.uq.edu.au/shenht/UQ_VIDEO/.

Authors' contributions
The first author YL contributed the main idea of this work and conducted the experiments. The second author LH helped in supervising the experiments and writing of the paper. The third author KX helped conduct the experiments. The fourth author JL refined the main idea of this work and polished the writting. All authors read and approved the final manuscript.

Competing interests
The authors declare that they have no competing interests.

Publisher's Note

Author details
[1]College of Computer Science and Technology, Jilin University, Jilin, China. [2]School of Economics and Management, Changchun University of Technology, Jilin, China. [3]State Key Lab of Software Development Environment, Beihang University, Beijing, China.

Received: 7 November 2018 Accepted: 30 January 2019
Published online: 05 March 2019

References

1. Y. Cai, L. Yang, W. Ping, F. Wang, T. Mei, X.-S. Hua, S. Liu, in *19th ACM international conference on Multimedia*. Million-scale near-duplicate video retrieval system (ACM, Scottsdale, 2011), pp. 837–838
2. J. Song, Y. Yang, Z. Huang, H. T. Shen, R. Hong, in *ACM international conference on Multimedia*. Multiple feature hashing for real-time large scale near-duplicate video retrieval (ACM, Scottsdale, 2011), pp. 423–432
3. J. L. Bentley, Multidimensional binary search trees used for associative searching. Commun. ACM. **18**, 509–517 (1975)
4. J. He, J. Feng, X. Liu, T. Cheng, T.-H. Lin, H. Chung, S.-F. Chang, in *IEEE conference on computer vision and pattern recognition*. Mobile product search with bag of hash bits and boundary reranking (IEEE, Providence, 2012), pp. 3005–3012
5. X. Liu, Z. Li, C. Deng, D. Tao, Distributed Adaptive Binary Quantization for Fast Nearest Neighbor Search. IEEE Trans. Image Process. **26**(11), 5324–5336 (2017)
6. J. Song, T. He, L. Gao, X. Xu, H. Shen, in *AAAI Conference on Artificial Intelligence*. Deep region hashing for efficient large-scale instance search from images (AAAI, New Orleans, 2018), pp. 402–409
7. M. S. Charikar, in *ACM Symposium on Theory of Computing*. Similarity estimation techniques from rounding algorithms (ACM, Montreal, 2002), pp. 380–388
8. W. Liu, J. Wang, R. Ji, Y.-G. Jiang, S.-F. Chang, in *IEEE conference on computer vision and pattern recognition*. Supervised hashing with kernels (IEEE, Providence, 2012), pp. 2074–2081
9. X. Liu, Y. Mu, B. Lang, S. Chang, Mixed image-keyword query adaptive hashing over multilabel images. ACM Trans. Multimedia Comput. Commun. Appl. **10**(2), 1–21 (2014)
10. Y. Mu, S. Yan, in *AAAI Conference on Artificial Intelligence*. Non-metric locality-sensitive hashing (AAAI, Atlanta, 2010), pp. 539–544

11. Y. Gong, S. Kumar, H. A. Rowley, S. Lazebnik, in *IEEE conference on computer vision and pattern recognition*. Learning binary codes for high-dimensional data using bilinear projections (IEEE, Portland, 2013), pp. 484–491
12. J.-P. Heo, Y. Lee, J. He, S.-F. Chang, S.-E. Yoon, in *IEEE conference on computer vision and pattern recognition*. Spherical hashing (IEEE, Providence, 2012), pp. 2957–2964
13. B. Kulis, K. Grauman, in *IEEE International Conference on Computer Vision*. Kernelized locality-sensitive hashing for scalable image search (IEEE, Kyoto, 2009), pp. 2130–2137
14. C. Deng, Z. Chen, X. Liu, X. Gao, D. Tao, Triplet-based deep hashing network for cross-modal retrieval. IEEE Trans. Image Process. **27**(8), 3893–3903 (2018)
15. E. Yang, C. Deng, W. Liu, X. Liu, D. Tao, X. Gao, in *AAAI Conference on Artificial Intelligence*. Pairwise relationship guided deep hashing for cross-modal retrieval (AAAI, San Francisco, 2017), pp. 1618–1625
16. Y. Gong, S. Lazebnik, Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. IEEE Trans. Pattern Anal. Mach. Intell. **35**(12), 2916–2929 (2013)
17. W. Liu, C. Mu, S. Kumar, S.-F. Chang, in *Advances in neural information processing systems*. Discrete graph hashing (Curran Associates, Inc., Montreal, 2014), pp. 3419–3427
18. M. Norouzi, D. J. Fleet, in *IEEE Conference on Computer Vision and Pattern Recognition*. Cartesian k-means (IEEE, Portland, 2013), pp. 2938–2945
19. X. Liu, L. Huang, C. Deng, B. Lang, D. Tao, Query-Adaptive Hash Code Ranking for Large-Scale Multi-View Visual Search. IEEE Trans. Image Process. **25**(10), 4514–4524 (2016)
20. X. Liu, J. He, S. Chang, Hash Bit Selection for Nearest Neighbor Search. IEEE Trans. Image Process. **26**(11), 5367–5380 (2017)
21. C. Deng, H. Deng, X. Liu, Y. Yuan, Adaptive multi-bit quantization for hashing. Neurocomputing. **151**(1), 319–326 (2015)
22. C. Deng, X. Liu, Y. Mu, J. Li, Large-scale multi-task image labeling with adaptive relevance discovery and feature hashing. Signal Process. **112**(C), 137–145 (2015)
23. L. Cao, Z. Li, Y. Mu, S. Chang, in *ACM international conference on Multimedia*. Submodular video hashing: a unified framework towards video pooling and indexing (ACM, Nara, 2012), pp. 299–308
24. H. Zhang, M. Wang, R. Hong, T. Chua, in *ACM international conference on Multimedia*. Play and rewind: optimizing binary representations of videos by self-supervised temporal hashing (ACM, Amsterdam, 2016), pp. 781–790
25. J. Song, H. Zhang, X. Li, L. Gao, M. Wang, R. Hong, Self-supervised video hashing with hierarchical binary auto-encoder. IEEE Trans. Image Process. **27**(7), 3210–3221 (2018)
26. K. Xia, Y. Ma, X. Liu, Y. Mu, L. Liu, in *ACM international conference on Multimedia*. Temporal binary coding for large-scale video search (ACM, Mountain View, 2017), pp. 333–341
27. B. Wang, X. Liu, K. Xia, K. Ramamohanarao, D. Tao, in *Pacific Rim Conference on Multimedia*. Random angular projection for fast nearest subspace search (Springer, Hefei, 2018), pp. 15–26
28. X. Liu, C. Deng, B. Lang, D. Tao, X. Li, Query-Adaptive Reciprocal Hash Tables for Nearest Neighbor Search. IEEE Trans. Image Process. **25**(2), 907–919 (2016)
29. Q. Fu, X. Han, X. Liu, J. Song, C. Deng, in *International Joint Conferences on Artificial Intelligence*. Binary quantization for joint multiple indexing (IJCAI, Stockholm, 2018), pp. 2114–2120
30. N. Kumar, R. Rawat, S. C. Jain, in *IEEE International Conference on Infocom Technologies and Optimization*. Bucket based data deduplication techniquefor big data storage system (IEEE, Noida, 2016), pp. 267–271
31. D. Moise, D. Shestakov, G. Gudmundsson, L. Amsaleg, in *ACM International conference on multimedia retrieval*. Indexing and searching 100M Images with Map-Reduce (ACM, Dallas, 2013), pp. 17–24
32. Y. Li, K. Xia, in *ACM International Conference on Internet Multimedia Computing and Service*. Fast video deduplication via locality sensitive hashing with similarity ranking (ACM, Xi'an, 2016), pp. 94–98
33. X. Liu, J. He, C. Deng, B. Lang, in *IEEE conference on computer vision and pattern recognition*. Collaborative hashing (IEEE, Columbus, 2014), pp. 2147–2154
34. K. He, F. Wen, J. Sun, in *IEEE conference on computer vision and pattern recognition*. K-means hashing: an affinity-preserving quantization method for learning binary compact codes (IEEE, Portland, 2013), pp. 2938–2945
35. J. Cheng, C. Leng, J. Wu, H. Cui, H. Lu, in *IEEE Conference on Computer Vision and Pattern Recognition*. Fast and accurate image matching with cascade hashing for 3d reconstruction (IEEE, Columbus, 2014), pp. 4321–4328
36. J. He, S. Kumar, S.-F. Chang, in *International Conference on Machine Learning*. On the difficulty of nearest neighbor search (ICML, Edinburgh, 2012), pp. 1–8
37. M. Norouzi, A. Punjani, D. J. Fleet, in *IEEE Conference on Computer Vision and Pattern Recognition*. Fast search in hamming space with multi-index hashing (IEEE, Providence, 2012), pp. 3108–3115
38. G. Ye, D. Liu, J. Wang, S. F. Chang, in *IEEE international conference on computer vision*. Large-scale video hashing via structure learning (IEEE, Sydney, 2013), pp. 2272–2279
39. X. Wu, A. G. Hauptmann, C.-W. Ngo, in *ACM international conference on Multimedia*. Practical elimination of near-duplicates from web video search (ACM, Augsburg, 2007), pp. 218–227
40. M. Slaney, Y. Lifshits, J. He, Optimal parameters for locality-sensitive hashing. Proc. IEEE. **100**(9), 2604–2623 (2012)