TWO-STAGE SPARSE GRAPH CONSTRUCTION USING MINHASH ON MAPREDUCE

Liang-Chi Hsieh[⋆] *Guan-Long Wu*[⋆] *Wen-Yu Lee*[⋆] *Winston Hsu*[†]

[⋆] Graduate Institute of Networking and Multimedia, National Taiwan University, Taipei, Taiwan
[†]Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan

ABSTRACT

Image graph attracts attention from researchers due to the empirical success of graph based semi-supervised learning (SSL) methods and tasks such as image clustering, image navigation. Despite its simple structure, overwhelming scale of online images makes image graph construction a difficult problem. The challenge lies in time-consuming computation, and the difficulty of storing, processing the resulted graphs of huge size. We propose a novel method of image graph construction on MapReduce for large-scale data. The method consists of two stages: the first stage separates images into overlapping groups called image pools by using hash method, and the second computes pairwise similarities for pairs of images that are grouped into common pools. Both stages are performed on MapReduce. Our experiments on large-scale data show that the proposed method generates more sparse image graphs that reserve same or improved accuracy when comparing with previous method.

Index Terms— Image graph, hash, sparse graph

1. INTRODUCTION

Image graph attracts much attention from researchers due to the empirical success of graph based semi-supervised learning (SSL) methods [6][5][17]. Image graph is also helpful in other tasks such as image clustering, image navigation. Image graph uses the metaphor of nodes and links between nodes to represent images and relations among images.

Despite the simplicity of the structure of image graph, overwhelming scale of available image data gives challenge to the construction of image graph. Specially, the challenge lies in two aspects: time-consuming computation for processing large-scale image data, and the difficulty of storing, processing and retrieving the huge amount of resulted image graphs. In this research, we propose a novel method of image graph construction for large-scale image data which leverages the parallel computing power of MapReduce [4] and uses hash method to produce more sparse graph that reserves accuracy.

The proposed method consists of two stages as shown in Figure 1. Initially, the first stage uses hash method to separate images into coarse groups called image pools. We use MinHash [1] to fit our sparse image representation and similarity measure. For improving the recall of hashing results, a multiple-table approach of MinHash is applied to generate overlapping image pools. In the second stage, the pairs of images that have been hashed into common image pools, are computed for pairwise similarities to construct image graph. Both two stages are performed on MapReduce platform. Since most unimportant links in graph are filtered in hashing process, we find that the graphs generated by our method are more sparse (i.e., occupying less storage space) but reserve about the same or improved accuracy.

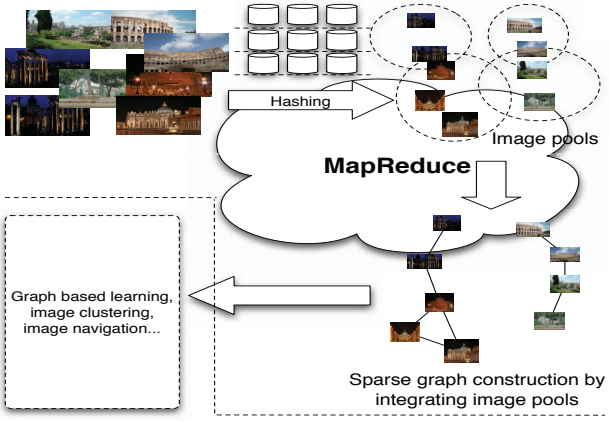

Fig. 1. The proposed two-stage sparse image graph construction method. First stage initially divides images into overlapping subsets called image pools. In the second stage, pairs of images divided into same image pools are computed for pairwise similarity to construct image graph. Both stages are performed on MapReduce platform to scale up for large-scale images. Resulted sparse graph can be used in graph based learning, image clustering and navigation.

Our main contribution in this work is to propose a novel method of image graph construction which generates more sparse image graph that reserves expected accuracy when comparing with previous method. To demonstrate our method, we evaluate it on a large-scale data set of 550k images and show the effectiveness and efficiency.

The paper is organized as follows. In Section 2 we review related works of graph construction. We introduce the two-stage graph construction method in Section 3. The experiments for demonstrating the proposed method on large-scale data sets are showed in Section 4. We will give our conclusion in Section 5.

2. RELATED WORKS

Compared to label inference in graph based SSL methods, crucial graph construction phase attracts less attention until recently [11] [16]. Among proposed graph construction methods, k-Nearest Neighbor (k-NN) and ϵ-neighborhood are commonly used in graph based SSL methods. Jebara et al. [11] argued the importance of regular graphs for graph based SSL methods and proposed graph construction method using b-matching. While the graph construc-

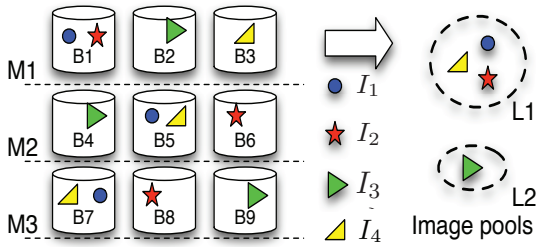

Fig. 2. Illustration of creating image pool by using MinHash technique. Four images I_1, I_2, I_3, I_4 are hashed into buckets after applying three MinHash tables $M1$, $M2$ and $M3$, respectively. Because I_1 and I_2 are hashed into same bucket in $M1$ table, they are put in same image pool. Later they are computed pairwise similarity. On the contrary, I_3 between I_1, I_2 or I_4 will not be computed.

tion using *b*-matching shows empirical advantage in [11], theoretical guarantees for the advantages of *b*-matching are not provided as suggested in [11] [16]. Furthermore, high computational complexity of *b*-matching is one of the major concerns, especially when applying on large scale data.

The simple but powerful parallel computing provided by MapReduce model [4] has proliferated in many domains from text processing to machine learning since the need to learn for growing large scale data becomes urgent. Based on the previous work to scale up all pairs similarity search [7], Elsayed et al. [8] proposed to compute pairwise document similarity in large data collections using MapReduce. Their approach uses two MapReduce jobs. The first job parses tokens in documents and generates inverted lists. The second job calculates partial pairwise similarities for pairs of documents in each list and sums those partial similarities for each pair to obtain final pairwise document similarity.

Lin introduced three MapReduce algorithms in [12] to explore the problem of computing pairwise similarity on documents. The algorithms are basically based on inverted list approach similar to [8], but add approximation strategies to improve the efficiency of graph construction. However, those algorithms can not be used to improve effectiveness of resulted graphs. In addition, the issue of graph sparseness has not been discussed in [12]. Graph sparseness impacts graph learning in effectiveness and efficiency.

3. TWO-STAGE GRAPH CONSTRUCTION USING MINHASH

We propose two-stage graph construction using MinHash in this section. The idea is motivated by [14] that proposed to cluster high dimensional data. Similar to [14], we also divide images into overlapping subsets and perform subsequent computation within each subset. However, we consider the scalability issue in both stages.

3.1. Creating Image Pools by Using MinHash

We deploy LSH based approach to efficiently divide images into overlapping subsets we call *image pools*. LSH [10] provides an efficient and effective technique for finding near neighbors in large scale data set. Many LSH schemes are known for different distance or similarity measures. In this work, we represent images as high dimensional feature vectors by using the "bag of words" model. Since those features are very sparse, we decide to use Jaccard coefficient as similarity measure oft images.

MinHash [1] is a LSH scheme for using Jaccard coefficient as similarity measure. Under the "bag of words" model, it assigns image pairs into same hashing bucket with the probability proportional to the overlapping between the features of images. In fact, MinHash can be thought as a probabilistic clustering method that clusters two images I_i and I_j into same cluster with the probability $S(I_i, I_j) = \frac{|W_{I_i} \cap W_{I_j}|}{|W_{I_i} \cup W_{I_j}|}$ that equals to overlap similarity of features (W_{I_i} and W_{I_j}) in their features. On purpose of computing the probability $S(I_i, I_j)$, MinHash method randomly permutes the set of features of images. For each image I_i, the hash value $h(I_i)$ is the index of the first features of I_i in the permutation.

In order to tackle the problem of performing MinHash on large scale images, we apply MapReduce based MinHash proposed by Das et al. [3] for online collaborative filtering. Its main idea is to use random seed values to replace random permutations in MinHash, because it is not feasible to generate random permutations over large word sets to compute MinHash values under a distributed environment.

Images applied MinHash are divided into non-overlapping buckets. Because the high precision and low recall of MinHash method, simply computing pairwise similarity for images within common buckets will generate graphs that are too sparse to be useful in tasks such as learning.

In order to improve that, we can repeat to perform MinHash on images and see each MinHash performing as a hashing table. Although the buckets in a hashing table are non-overlapping, overlapping can be happened between buckets from different hashing tables. By joining overlapping buckets from different hashing tables, we create the overlapping image subsets called image pools.

For example, as illustrated in Figure 2, given 4 images I_1, I_2, I_3 and I_4, they are hashed into buckets $\{M1_{B1}, M2_{B5}, M3_{B7}\}$, $\{M1_{B1}, M2_{B6}, M3_{B8}\}$, $\{M1_{B2}, M2_{B4}, M3_{B9}\}$ and $\{M1_{B3}, M2_{B5}, M3_{B7}\}$ after applying 3 MinHash tables $M1, M2, M3$, respectively. It is shown that I_1, I_2 are hashed into same bucket $M1_{B1}$ and I_1, I_4 are hashed into same bucket $M3_{B7}$. By joining $M1_{B1}$ and $M3_{B7}$ overlapping with I_1, we create image pool $L1$.

By tuning the times of performing MinHash, we can adjust the sparsity and accuracy of generated graphs. Related evaluations are shown in Section 4.

When we assume that in each MinHash process two similar images and two dissimilar images are hashed into same bucket with probability p (positive result) and q (negative result), respectively. Assume N_h is the number of hashing tables used. When N_h increases, $(1-p)^{N_h}$ getting smaller that indicates better performance, but $(1-q)^{N_h}$ also getting smaller that means worse performance. Generally, for a LSH scheme, similar data are more likely hashed into same bucket ($p > (1-p)$) and dissimilar data are fallen into different buckets with higher probability ($(1-q) > q$). Thus, we can expect that by applying multiple hash tables to create image pools, the performance of resulted graph becomes better and then worse as the the number of tables N_h increases. Because more computation time is needed for more hash tables, it suggests that we have trade off between graph effectiveness and efficiency in our two-stage graph construction method.

Chum et al. [2] proposed using MinHash to compute approximate similarity in solving the problem of near duplicate image and video-shot detection. It is noteworthy that they also use multiple hash tables to improve the precision of image retrieval. For example, if there are m hash tables, the retreival method estimates similarity

1: **procedure** Reduce ($\langle i_i, i_j \rangle$, $[w_{ij1}, w_{ij_2}....]$)
2: Input:
Pair of image ids $\langle i_i, i_j \rangle$,
List of weight products $[w_{ij1}, w_{ij_2}....]$
3: **if** InCommonPools($\langle i_i, i_j \rangle$) = **true then**
4: $sim = 0$
5: **for all** $w_{ij} \in [w_{ij1}, w_{ij_2}....]$ **do**
6: $sim = sim + w_{ij}$
7: **end for**
8: EMIT($\langle i_i, i_j \rangle$, sim)
9: **end if**

Fig. 3. Pseudo-code of computing pairwise similarity on MapReduce for integrating image pools (only reduce function here). The details about map function, such as the computation of $[w_{ij1}, w_{ij_2}....]$, can refer to [8].

only for image pairs that have been fallen into same buckets for all m hash tables. However, we use multiple hash tables to improve recall and reduce the sparness of generated image graphs. In the other words, our method estimates similarity for image pairs that are fallen into at least a same bucket in any of the m hash tables.

3.2. Computing Pairwise Similarity by Integrating Image Pools

Our pairwise similarity computation method is motivated by the MapReduce based approach in [8] that converts text documents into inverted lists and computes pairwise similarity for documents. We apply [8] to compute image similarity but extend it by integrating image pools to filter noisy links of graph.

Most simple approach to integrate image pools is to directly append it into inverted list. But it is not difficult to recognize that the bottleneck would be accompanying intermediate output and heavy disk operation, especially for a distributed model like MapReduce. Growing intermediate output between mapper and reducer functions downgrades computing efficiency as applying on larger data. Thus we propose to integrate image pools into pairwise similarity computation in the reduce step of [8]. The algorithm is shown in Figure 3, but omits the unmodified map step since it is a similar step as [8] and [12]. At the line 2 in Figure 3, the function *InCommonPools* checks if the pair of images is divided in common image pool. If so, then their products of feature weights are sum up and final similarity is emitted.

4. EXPERIMENTS

We perform experiments to validate our proposed two-stage graph construction on two data sets:

Flickr550 data set: We take the large-scale image data set Flickr550 [18] as our experiment data that contains 540321 images.

Flickr11k data set: Flickr11k [9] is a data set consisting of 11282 medium resolution (500x360) images. This is a subset of Flickr550 data set. It consists of 1282 ground truth images in 7 query categories [1] and 10k images randomly sampled from Flickr550.

Those images are represented in both visual and textual high dimensional features. We deploy visual word (VW) as visual features to compute similarity of images. In order to generate VW, we use Difference-of-Gaussian (DoG) to detect feature points and describe them with SIFT [13]. Detected descriptors are then quantized into 10K clusters using k-means method. The centroids of clusters are defined as VWs. A feature point in image is assigned to a VW that is nearest to the feature point's descriptor. For textual representation, we adopt web-based kernel function [15] and Google search engine for perform query expansion to represents image in textual feature of 91004 dimension.

We implement MapReduce algorithms in this work in Java on Hadoop platform that is an open source implementation of MapReduce programming model. Those experiments are run on two middle Hadoop clusters that consist of 18 and 24 commodity machines, respectively. Version 0.20.1 Hadoop is used on both clusters.

4.1. Result and Discussion

We generate visual and textual graphs for Flickr11k using both one-stage method described in [8] (ONE) and our two-stage image graph construction method (TWO) approach. Graphs are measured for performance by using a retrieval based approach. Given a graph, each image in graph is taken as a query and its top k ($k = 1000$ in this work) near neighbors are retrieved as a ranking list. Average precision (AP) can be computed for the ranking list. By averaging AP values for all images in graph, we obtain MAP performance for the graph. We use tf-idf threshold to filter out image feature weights before computing pairwise image similarity. The tf-idf based feature filtering helps removing insignificant features and thus decreases graph size for fitting disk capacity of computing clusters.

First we want to compare the performance of graphs generated by ONE and TWO methods. Table 1 shows the comparison by varying t, the number of MinHash tables used by TWO. Observed from the table, using more hash tables helps MAP at the beginning. While MAP downgrades after it peaks at $t = 6$, TWO still generates graphs with competent performance.

By using MinHash method to create image pools and computing pairwise similarity within each image pool, the graphs of TWO are much more sparse than ONE. The advantage of TWO is significant as we take sparseness, graph size and MAP into account at the same time. For example, under tf-idf threshold 0.0003, the Flickr11k visual graph of ONE occupies 280MB disk space. The graph with same configuration generated by TWO using 6 MinHash tables only accounts for 27MB. But TWO graph obtains a better MAP 0.23 than 0.21 of ONE. In order to represent sparseness of graph, we define it as e/n^2 where n is the number of images and e is the number of edges. The sparseness of the ONE graph and TWO graph is 0.31 and 0.03, respectively. The result suggests that graphs generated by our TWO method are more sparse but have higher performance than ONE method. While it is feasible to increase td-idf cutting threshold to increase sparseness of graphs generated by ONE, MAP of graphs will decrease at the same time. We illustrate the relation between graph sparseness and MAP in Figure 4, for graphs of ONE and TWO. The curve of ONE in the figure varies by changing tf-idf cutting threshold; increasing threshold raises graph sparseness but decreases MAP. The curve of TWO varies by changing the number of MinHash tables; increasing the number of table in a limited range lowers sparseness but augments MAP. Beyond a particular number of tables, using more tables only makes graph more dense but has no contribution to MAP. Observed from Figure 4, the graphs of TWO are much more sparse than ONE at a same MAP. On the other hand, when they are similar in sparseness, the graphs of TWO have higher MAP.

For efficiency, while our methods generate more intermediate outputs between MapReduce jobs, the scale of Flickr11k does not manifest the difference in computing time between ONE and TWO. For comparing efficiency, we compare the construction time of ONE

[1] colosseum, eiffel tower, golden, torre pendente di pisa, starbucks, tower bridge, triomphe

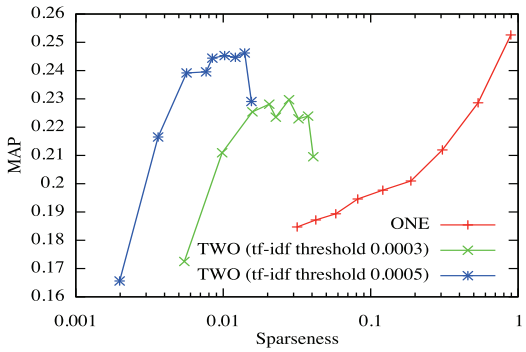

Fig. 4. The relation between MAP and graph sparseness (from most sparse 0 to most dense 1) for ONE and TWO. We show the curves of graphs generated with varying hashing table number (t = 1 ˜9) using our method for Flickr11k visual features under two thresholds. The graphs of ONE curve are generated by varying tf-idf cut-off thresholds.

	ONE	TWO, $t = 2$	TWO, $t = 4$	TWO, $t = 6$	TWO, $t = 8$
MAP	0.21	0.21	**0.23**	**0.23**	0.22

Table 1. MAP comparison between Flickr11k visual graphs generated by ONE method and TWO under same tf-idf threshold. t is the number of MinHash tables used in TWO.

and TWO method on Flickr550 visual features. Because the high efficiency of MinHash technique, the time of applying MinHash on the two data sets can be ignored, compared to graph construction time. We show the results of TWO (t = 8) and ONE for Flickr550 visual feature at tf-idf threshold 0.001 in Table 2. Although TWO method adds extra overhead and has slightly longer computing time, it returns better MAP and smaller graph size.

5. CONCLUSION

Image graph attracts attention from researchers in recent years due to the success of graph based learning methods. Image graph is also helpful in image clustering, image navigation applications. Although it is easy to construct image graph for dozens of images, the overwhelming scale of online images makes image graph construction a non-trivial problem. In this work, we propose a two-stage graph construction method based on MapReduce. We demonstrate the effectiveness and efficiency of the proposed method on two large scale image data sets. The experiment results show that the proposed method can generate more sparse image graph (i.e., occupying less storage space) that still has about the same or even higher performance when comparing with baseline graph. The smaller the generated image graph, the less the load on later learning method to process the graph.

Method	Load	Inverted list	Pairwise	MAP	Graph size
ONE	0 sec.	1137 sec.	859 sec.	0.029	7270MB
TWO, t = 8	2046 sec.	1080 sec.	645 sec.	**0.031**	**74MB**

Table 2. Computation time of different stages for ONE and TWO (t = 8) methods on Flickr550 visual graph under same tf-idf threshold. *Load* is the step to load image pools. While TWO method brings overhead of efficiency when loading image pools, it creates the graph with much smaller size and higher MAP.

6. REFERENCES

[1] A. Broder. On the resemblance and containment of documents. In *SEQUENCES*, pages 21–29, 1997.

[2] O. Chum, J. Philbin, M. Isard, and A. Zisserman. Scalable near identical image and shot detection. In *Proceedings of the 6th ACM international conference on Image and video retrieval*, 2007.

[3] A. S. Das, M. Datar, A. Garg, and S. Rajaram. Google news personalization: scalable online collaborative filtering. In *WWW*, pages 271–280, 2007.

[4] J. D. et al. Mapreduce: Simplified data processing on large clusters. In *OSDI'04*, 2004.

[5] J. L. et al. Video search re-ranking via multi-graph propagation. In *ACM Multimedia*, 2007.

[6] J. T. et al. Structure-sensitive manifold ranking for video concept detection. In *Proceedings of the 15th international conference on Multimedia*, pages 852–861, 2007.

[7] R. J. B. et al. Scaling up all pairs similarity search. In *WWW*, pages 131–140, 2007.

[8] T. E. et al. Pairwise document similarity in large collections with mapreduce. In *ACL-08: HLT*, pages 265–268, 2008.

[9] Y.-H. K. et al. Query expansion for hash-based image object retrieval. In *ACM Multimedia*, pages 65–74, 2009.

[10] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *STOC*, pages 604–613, 1998.

[11] T. Jebara, J. Wang, and S.-F. Chang. Graph construction and b-matching for semi-supervised learning. In *ICML*, pages 441–448, 2009.

[12] J. Lin. Brute force and indexed approaches to pairwise document similarity comparisons with mapreduce. In *SIGIR*, pages 155–162, 2009.

[13] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60 (2):91 – 110, 2004.

[14] A. McCallum, K. Nigam, and L. H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *ACM SIGKDD*, pages 169–178, 2000.

[15] M. Sahami and T. D. Heilman. A web-based kernel function for measuring the similarity of short text snippets. In *Proc. WWW*, 2006.

[16] P. P. Talukdar. Topics in graph construction for semi-supervised learning. Technical report, University of Pennsylvania, 2009.

[17] C. Wang, F. Jing, L. Zhang, and H.-J. Zhang. Image annotation refinement using random walk with restarts. In *ACM Multimedia*, pages 647–650, 2006.

[18] Y.-H. Yang, P.-T. Wu, C.-W. Lee, K.-H. Lin, and W. H. Hsu. Contextseer: Context search and recommendation at query time for shared consumer photos. In *ACM Multimedia*, 2008.