Learning Vocabulary-Based Hashing with AdaBoost

Yingyu Liang, Jianmin Li, and Bo Zhang

State Key Laboratory of Intelligent Technology and Systems
Tsinghua National Laboratory for Information Science and Technology
Department of Computer Science and Technology, Tsinghua University
Beijing 100084, China
`liangyy08@mails.tsinghua.edu.cn`,
`{lijianmin,dcszb}@mail.tsinghua.edu.cn`

Abstract. Approximate near neighbor search plays a critical role in various kinds of multimedia applications. The vocabulary-based hashing scheme uses vocabularies, i.e. selected sets of feature points, to define a hash function family. The function family can be employed to build an approximate near neighbor search index. The critical problem in vocabulary-based hashing is the criteria of choosing vocabularies. This paper proposes a approach to greedily choosing vocabularies via Adaboost. An index quality criterion is designed for the AdaBoost approach to adjust the weight of the training data. We also describe the parallelized version of the index for large scale applications. The promising results of the near-duplicate image detection experiments show the efficiency of the new vocabulary construction algorithm and desired qualities of the parallelized vocabulary-based hashing for large scale applications.

1 Introduction

Approximate nearest neighbor search plays a critical role in various kinds of multimedia applications, including object recognition [13], near-duplicate image detection [10], content-based copy detection [11,16]. In such applications, typically, the multimedia objects are represented as sets of elements (e.g., local feature points), between which the similarity can be evaluated via searching the nearest neighbors of each element.

The recent explosion of multimedia data has led the research interest into large scale multimedia scene. The various typical approximate near neighbor search algorithms, such as ANN [4] and LSH[5,8], show high performance in relatively small datasets, but do not fit in the large scale scene. For example, the popular Euclidean locality sensitive hashing based on p-stable distributions (E2LSH)[5] typically requires hundreds of bytes for each point. Also, instead of all points in the buckets, it performs a refinement step to return only those within a distance threshold, which requires loaded data in the memory. These shortcomings prevent it from usage in large datasets.

The bag-of-features (BOF) image representation [17] is introduced in this context. Each feature point in the dataset is quantized by mapping to the ID of the nearest one in a selected set of feature points called a visual vocabulary. The vector quantization approach can be interpreted as an approximate near neighbor search: the space is partitioned into Voronoi cells by the vocabulary, and points are treated as neighbors of each other if they lie in the same cell. The BOF approach can deal with large scale datasets for its efficient and space-saving property. However, it is an approximation to the direct matching of individual feature points and somewhat decreases the performance [10].

The vocabulary-based hashing scheme [12] combines the merits of BOF and LSH. Vocabularies are employed to define a hash function family in which each function maps an input point to the ID of the nearest one in the corresponding vocabulary. The function family is incorporated into the locality sensitive hashing scheme in [9] to build an index for approximate near neighbor search. This approach shows better performance than BOF and LSH, and it is efficient for large databases. In this vocabulary-based hashing scheme, the vocabularies define the hashing functions and thus determine the index, so they play a key role and should be carefully designed. The vocabulary construction algorithm in [12] first generates random vocabularies and then selects from them according to two criteria. However, it is time-consuming since it must generate a large amount of random vocabularies to select effective vocabularies.

In this paper, a new approach utilizing AdaBoost [7] is proposed for the vocabulary construction in vocabulary-based hashing. An index quality criterion is designed for AdaBoost to adjust the weight of the training data. We also describe the parallelized version of the index for large scale applications. Near-duplicate image detection experiments are carried out to demonstrate the effectiveness and efficiency of the approach. The results show that the new vocabulary construction algorithm is significantly more efficient than that in [12], and the parallelized vocabulary-based hashing shows desired qualities for the large scale scene.

This paper is organized as follows. The vocabulary-based hashing index is briefly reminded in Section 2. Section 3 presents the proposed vocabulary construction algorithm and Section 4 describes the parallelization. Experimental results are provided in Section 5. Section 6 concludes the paper.

2 Vocabulary-Based Hashing

In this section we briefly describe the vocabulary-based hashing scheme proposed in [12].

Denote a hash function family mapping a domain S into U as $\mathcal{H} = \{h : S \rightarrow U\}$. [12] propose to use feature point vocabularies to define hash functions by partitioning the space into Voronoi cells. Formally, A hash function $h \in \mathcal{H}$ is defined as

$$h(q) = \arg \min_{0 \leq i < t} D(q, w_h^i), w_h^i \in V_h$$

where

$$V_h = \{w_h^i, 0 \leq i < t\}$$

is a vocabulary associated with h, t is the size of the vocabulary and $D(q, w)$ is the Euclidean distance between points q and w.

Here we remind the hashing index scheme using a given function family [9]. First for a given parameter k, define a function family $\mathcal{G} = \{g : S \to U^k\}$ such that $g(p) = (h_1(p), \ldots, h_k(p))$, where $h_i \in \mathcal{H}$. Then for a given parameter L, choose L functions $g_1, \ldots, g_L$ from $\mathcal{G}$. During the construction of the index, each data point p is stored in the buckets $g_j(p)$, for $j = 1, \ldots, L$. To find neighbors for a query point q, search all buckets $g_1(q), \ldots, g_L(q)$ and return all the points encountered. Thus, the functions $g_1, \ldots, g_L$ define a hashing index and different hashing function family $\mathcal{H}$ leads to different index. The vocabulary-based hashing index is constructed by employing the vocabulary-based hash functions. For simplicity, we call $V_g = (V_{h_1}, \ldots, V_{h_k})$ a vocabulary associated with $g = (h_1, \ldots, h_k)$ and let $V = (V_{g_1}, \ldots, V_{g_L})$.

3 Vocabulary Construction

As mentioned above, the vocabularies play a key role in the scheme. The basic idea for the vocabulary construction in [12] is to select vocabularies of best quality from randomly generated ones. It is time-consuming because sufficient amount of random vocabularies are required for selection. We propose an algorithm that utilizes AdaBoost [7] to speed up the construction. The AdaBoost approach needs a criterion for representing the quality of a point being indexed to adjust the weight of the training data. So the first subsection focus on designing the criterion. Then we describe the vocabulary construction algorithm and provide an analysis of the AdaBoost approach in this context.

3.1 Index Quality Criterion

As noted in [12,10], a high-quality search index should return ground truth points and filter noise points with high probability at the same time. For example, in the typical application of similar image search, the retrieved neighbors are used for voting. Here the true positive neighbors can be regarded as useful information while the false positive neighbors bring noise into the voting. Thus we define the index quality of a point to be the signal/noise ratio of the returned neighbors if the point is used as a query. Formally, denote the dataset as $\mathcal{P}$. Suppose p and q are near neighbors if $D(p, q) < R$. Let

$$\begin{aligned} TP_g(q) &= \{p : g(p) = g(q), D(p, q) < R\} \\ T(q) &= \{p : D(p, q) < R\} \\ P_g(q) &= \{p : g(p) = g(q)\}. \end{aligned}$$

Assume there is only one true positive neighbor. Its possibility of being returned is $|TP_g(q)|/|T(q)|$, which can be used as the measure for information brought in. Further assume the weight of the noise brought in by one returned neighbor is w. We define the index quality of q in g to be

$$\hat{v}_g(q) = \frac{|TP_g(q)|/|T(q)| + 1}{w|P_g(q)| + 1} \tag{1}$$

where $|\cdot|$ is the number of points in the set. Note that better designs are possible but left for future work.

Here we discuss the setting of w in detail. We simplify the analysis by making the following assumptions. First, the noise brought in by n returned neighbors will counteract the useful information brought in by true neighbors and thus $w=1/n$. Second, the data points come from N_o multimedia objects (e.g., images or videos) and the returned neighbors scatter among these objects uniformly and independently. Then if two or more of the n returned points belong to the same object, the information of the true neighbors will be counteracted. Thus, we expect that the n noisy points belong to different multimedia objects with high probability:

$$\frac{\prod_{i=1}^{n-1}(N_o - i)}{N_o^{n-1}} > 1 - \epsilon.$$

Since $\ln(1+x) \approx x$ with small x, approximately we have,

$$\sum_{i=1}^{n-1}(\frac{i}{N_o}) < \epsilon.$$

Setting $\epsilon = 0.05$, we have

$$w = \frac{1}{n} \approx \sqrt{\frac{10}{N_o}}. \qquad (2)$$

The formula (1) and (2) define the criterion $\hat{v}_g(p)$. We tune it to fit the AdaBoost scheme as follows: if $\hat{v}_g(p)$ is among the smallest $|\mathcal{P}|/10$ ones, then $v_g(p) = -1$, indicating that p is not well-indexed in g and needs more emphasis; otherwise, $v_g(p) = 1$. Formally,

$$I_g(p,q) = \begin{cases} 1 & \text{if } \hat{v}_g(p) > \hat{v}_g(q) \\ 0 & \text{otherwise} \end{cases} \qquad (3)$$

$$v_g(p) = \begin{cases} -1 & \text{if } \sum_{q\in\mathcal{P}} I_g(p,q) < |\mathcal{P}|/10 \\ 1 & \text{otherwise} \end{cases} \qquad (4)$$

3.2 AdaBoost for Vocabulary Construction

Now we have the criterion $v_g(p)$ describing the quality of a point p indexed in the hash table g and thus can utilize AdaBoost in vocabulary construction. During the construction of $V = (V_{g_1}, \ldots, V_{g_L})$, we compute $\{v_{g_i}(p), p \in \mathcal{P}\}$ after V_{g_i} is constructed. Then the weights of the points are adjusted accordingly, emphasizing points with low index quality in the construction of $V_{g_{i+1}}$.

The algorithm is described in **ConstructVocabulary**$(\mathcal{P}, t, k, L, C)$: $\mathcal{P}$ is the training dataset; $\mathcal{W}$ is the weight for points in $\mathcal{P}$; t, k and L are index parameters; C is a parameter indicating the number of repetitions, typically $C = 10$.

We first briefly analyze the AdaBoost approach in the context of vocabulary construction, which is analogous to that in the classification context. During the

```
Procedure. ConstructVocabulary($(\mathcal{P}, t, k, L, C)$)
  1 For each $p \in \mathcal{P}$, assign the weight $\mathcal{W}_1(p) = 1/|\mathcal{P}|$
  2 For $i = 1$ to $L$
      1) $V_{g_i}$ = ConstructVg($(\mathcal{P}, \mathcal{W}_i, t, k, C)$)
      2) Compute $\{v_{g_i}(p), p \in \mathcal{P}\}$
      3) $\alpha_i$= ComputeAlpha($(\mathcal{W}_i, v_{g_i})$)
      4) For each $p \in \mathcal{P}$, $\mathcal{W}_{i+1}(p) = \mathcal{W}_i(p) \exp\{-\alpha_i v_{g_i}(p)\}$
      5) $Z_i = \sum_{p \in \mathcal{P}} \mathcal{W}_{i+1}(p)$
      6) For each $p \in \mathcal{P}$, $\mathcal{W}_{i+1}(p) = \mathcal{W}_{i+1}(p)/Z_i$
  3 Return $V = (V_{g_1}, \ldots, V_{g_L})$
```

analysis, we specify the subprocedures **ConstructVg** and **ComputeAlpha**. Let $N = |\mathcal{P}|, v_i(p) = v_{g_i}(p)$. We have

$$\begin{aligned}\mathcal{W}_{L+1}(p) &= \mathcal{W}_L(p) \frac{\exp\{-\alpha_L v_L(p)\}}{Z_L} \\ &= \mathcal{W}_1(p) \frac{\exp\{-\sum_{j=1}^{L} \alpha_j v_j(p)\}}{\prod_{j=1}^{L} Z_j} \\ &= \frac{\exp\{-\sum_{j=1}^{L} \alpha_j v_j(p)\}}{N \prod_{j=1}^{L} Z_j}.\end{aligned}$$

As $\sum_{p \in \mathcal{P}} \mathcal{W}_{L+1}(p) = 1$,

$$\begin{aligned}\prod_{j=1}^{L} Z_j &= \frac{1}{N} \sum_{p \in \mathcal{P}} \exp\Big\{-\sum_{j=1}^{L} \alpha_j v_j(p)\Big\} \\ &\geq \frac{1}{N} \sum_{p \in \mathcal{P}} \Big(1 - \sum_{j=1}^{L} \alpha_j v_j(p)\Big) \\ &= 1 - \frac{1}{N} \sum_{j=1}^{L} \alpha_j \sum_{p \in \mathcal{P}} v_j(p).\end{aligned}$$

Assume that the query shares the same distribution with the dataset, which leads to $E\big[v_j(q)\big] \approx \frac{1}{N} \sum_{p \in \mathcal{P}} v_j(p)$, then

$$\begin{aligned}\prod_{j=1}^{L} Z_j &\geq 1 - \sum_{j=1}^{L} \alpha_j \frac{1}{N} \sum_{p \in \mathcal{P}} v_j(p) \\ &\approx 1 - \sum_{j=1}^{L} \alpha_j E\big[v_j(q)\big].\end{aligned}$$

Thus, $1-\prod_{j=1}^{L} Z_j$ serves as a lower bound for $\sum_{j=1}^{L} \alpha_j E\big[v_j(q)\big]$, which indicates the quality of the index defined by $g_1, ..., g_L$. So we turn to minimize $\prod_{j=1}^{L} Z_j$. A greedy approach is adopted, i.e. incrementally minimize Z_j from $j=1$ to L. So the problem becomes

$$(g_j, \alpha_j)^* = \arg\min_{g,\alpha} \sum_{p\in\mathcal{P}} \mathcal{W}_j(p) \exp\{-\alpha v_g(p)\}$$

We greedily select g_j and then optimize α_j. An approximate solution of g_j will be

$$g_j = \arg\max_{g} \sum_{p\in\mathcal{P}} \mathcal{W}_j(p) v_g(p)$$

which leads to the following **ConstructVg**.

Subprocedure. ConstructVg$(\mathcal{P}, \mathcal{W}, t, k, C)$

```
1 Compute the mean $m$ of $\mathcal{P}$ and its bounding box $\mathcal{B}$, i.e. the minimum and
  maximum value in each dimension
2 For $i=1$ to $C$, $j=1$ to $k$
  Draw $t$ random points $p^i_{j,s} (0 \le s < t)$ within $\mathcal{B}$ uniformly and independently
3 Centralize to
      $w^i_{j,s} = p^i_{j,s} - \frac{1}{t}\sum_{s=0}^{t-1} p^i_{j,s} + m$
4 Let
      $V^i_{h_j} = \{w^i_{j,s}\}, V^i_g = (V^i_{h_1}, \ldots, V^i_{h_k})$
5 Return $V^i_g$ that maximizes $\sum_{p\in\mathcal{P}} \mathcal{W}(p) v_g(p)$
```

The objective Z_j becomes a function of α. It has nice analytical properties, and many algorithms exist for the optimization. We use Newton's algorithm in **ComputeAlpha**. During the experiments, we observe that setting the parameter $T=20$ will be sufficient for the procedure to converge.

Subprocedure. ComputeAlpha$(\mathcal{W}, v_g)$

```
1 $\alpha^{(0)} = 1.0$
2 For $i = 1$ to $T$
      $\alpha^{(i)} = \alpha^{(i-1)} + \frac{\sum_{p\in\mathcal{P}} \mathcal{W}(p) v_g(p) \exp\{-\alpha^{(i-1)} v_g(p)\}}{\sum_{p\in\mathcal{P}} \mathcal{W}(p) v_g^2(p) \exp\{-\alpha^{(i-1)} v_g(p)\}}$
3 Return $\alpha^{(T)}$
```

3.3 Implementation for Large Datasets

As in [12], we adopt a hierarchical approach for large datasets: the dataset is partitioned into t_1 subsets and vocabulary construction is performed for each subset. More specifically, during the vocabulary construction step, the dataset $\mathcal{P}$ is first clustered into t_1 points, which form the first level vocabulary $\widehat{V} = \{\widehat{w}^i, 0 \leq i < t_1\}$ for all $h \in \mathcal{H}$. Then we hash points on $\widehat{V}$ and each bucket forms a subset $\mathcal{P}_i$. The algorithm **ConstructVocabulary** uses each $\mathcal{P}_i$ as input dataset to construct vocabularies $V_i = (V_{i,g_1}, \ldots, V_{i,g_L}), V_{i,g} = (V_{i,h_1}, \ldots, V_{i,h_k}), V_{i,h} = \{w^s_{i,h}, 0 \leq s < t_2\}$, which form the second level. During the search step, we hash the query point on $\widehat{V}$, find which $\mathcal{P}_i$ it falls in, and use V_i to find its approximate near neighbors.

4 Parallelization

The vocabulary-based hashing can be parallelized naturally for the tables work in parallel. The parallelized version of the index is illustrated in Figure 1(L=2)[12]. The query is sent to each table and further forwarded to the corresponding bucket. Points in those buckets are then returned. The search time in vocabulary-based hashing consists of two parts: hashing on the first level $\widehat{V}$ needs $O(t_1)$ if brute-force search is adopted; hashing on V_i needs $O(t_2kL)$. After parallelization the second part of the search time is reduced to $O(t_2k)$. Also, each table can hold more points and thus the index can deal with larger datasets. Additionally, a single table can be further split into two or more tables which still work in parallel and thus can be deployed on machines without large memory. Our experiments in the next section show its benefits.

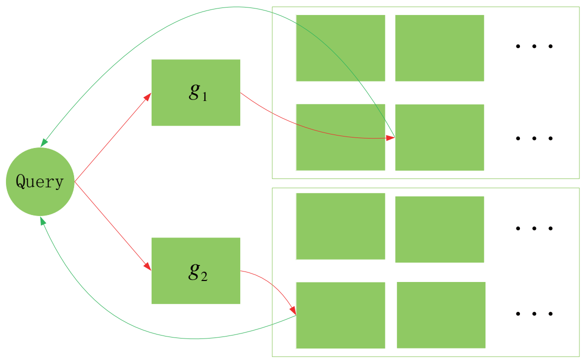

Fig. 1. Parallelized vocabulary-based hashing[12]

5 Experiments

5.1 Settings

We evaluate our index scheme in the typical application of near-duplicate image detection. The experiments are carried out on a Intel(R) Xeon(R) machine with 16GB of memory. The parallelized index is also distributed on other machines with Intel(R) Pentium(R) 4 CPU and 4G memory. The algorithms are implemented in C++.

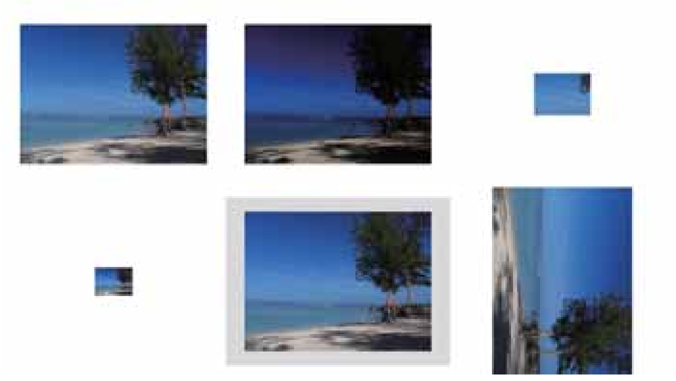

Fig. 2. Examples of near duplicates. From left to right, first row: original, change gamma to 0.5, crop 70%; second row: scale down by 5 times, frame, rotate 90°.

Datasets. We construct the vocabularies on $Flickr60k$ and evaluate on the $Holidays$ and $Flickr1M$ datasets [10,2]. The $Holidays$ (1491 images) is divided into 500 groups, each of which represents a distinct scene or object. The $Flickr60k$ (67714 images) and $Flickr1M$ (1 million images) are two distinct datasets downloaded arbitrarily from Flickr[1].

For vocabulary construction, we use a dataset $Flickr60k$ distinct from the testbed, in order to show more accurately the behavior in large scale scenes, where the dataset itself is too large, or it is updated incrementally so that we do not have the entire dataset at hand for the construction.

For evaluation, we construct the testbed similar to the web scale context from $Holidays$ and $Flickr1M$. The first image of each group in $Holidays$ is selected to form the query set. Transforms are applied to each query image and the generated duplicates are added into the $Flickr1M$ to form the test dataset. The transforms are similar to those in [11,14], and are implemented using ImageMagick[3]. They are listed below and the number in brackets next to each operation denotes the number of near-duplicate images generated.

SIFT [13] descriptors extracted from the images by the software in [2] are used in the experiments.

1. Exact duplicate [1].
2. Changing contrast [2]: (a) change contrast with default parameters in ImageMagick; (b) increase constrast by 3x.
3. Changing intensity [2]: intensity (a) decreased by 50%; (b) increased by 50%.
4. Changing gamma [2]: change gamma to (a) 0.5 or (b) 2.0.
5. Cropping [3]: crop the image by (a) 10% or (b) 50% or (c) 70%, preserving the center region.
6. Framing [1]: Add an outer frame to the image, where the size of the frame is 10% of the framed image.
7. Scaling [2]: scale the image down by (a) 2 or (b) 5 times.
8. Rotating [2]: Rotate image by (a) 90°, (b) 180°.
9. Inserting text [1]: insert the text at the center of the image.
10. Changing format [1]: change the image format JPEG to GIF.

Evaluation Measure. To evaluate the performance of the index, the near neighbor retrieved from the index are used to perform a vote on the images as in [10]. Note that in practice there is usually a post-verification step of the top n positions, especially in large scale scenes where the voting results need further refinements. So rate of true positives returned in the top n positions after voting (perf@n) serves as a suitable performance measure [15,10,6,12]. As there are 17 duplicate images, we choose perf@20 for our evaluation.

5.2 Results

Effectiveness. Figure 3 shows the effectiveness of the index while evaluating on the test data subsets of different sizes. VBH_Ada is the proposed approach with the index parameters $t_1 = 20000, t_2 = 2, k = 8, L = 8$. BOF is the bag-of-features approach with a codebook size of 20000. VBH_Ada outperforms BOF and shows better scalability since it is a refinement of BOF. VBH_Rs and VBH_Rn are vocabulary-based hashing with the same index parameters as VBH_Ada. VBH_Rs employs the construction approach in [12], generating 100 random vocabularies. VBH_Rn uses random vocabularies with words drawn from the bounding box of the dataset uniformly and independently at random. The better performance and scalability of VBH_Ada and VBH_Rs indicates that the construction algorithms do contribute to the effectiveness of the index.

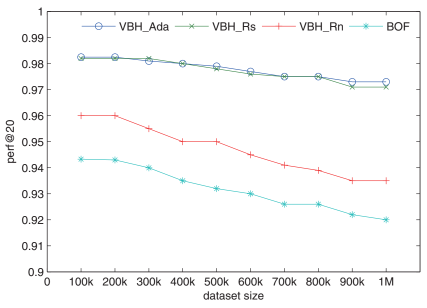

Fig. 3. Effectiveness

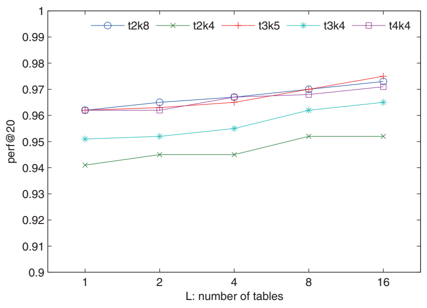

Fig. 4. Parameters

Parameters. Figure 4 shows the performance of vocabulary-based hashing on the 1M dataset with different index parameters t_1, t_2, k, L. All the settings use $t_1 = 20000$, and txky means $t_2 = x, k = y$. Indexes with similar number t_2^k shows similar performance and larger number leads to better results for larger t_2^k indicates finer partition of the feature space. Also the performance increase with table number L.

Time. The vocabulary construction time for the approach in [12] and our proposed approach is presented in table 1. Both approaches are applied on $Flickr60k$ to construct vocabularies with index parameters $t_1 = 20000, t_2 = 2, k = 8, L = 16$. The approach in [12] generates 100 random vocabularies for selection. The proposed approach consumes significantly less time since the AdaBoost method avoids the need for generating a large amount of random vocabularies. This makes it more practical in the large scale scene.

Table 1. Vocabulary construction time

Method	Time
[12]	154hr
proposed	21hr

Table 2. Feature extraction and search time per query (Flickr1M dataset)

Method	Feature extraction	Search
serial VBH	0.51s	8.97s
parallelized VBH	0.51s	1.27s
BOF	0.51s	9.16s

The search time for a query in the near-duplicate detection task is presented in table 2. Although the vocabulary-based hashing approach does more computation while searching the near neighbor, it filters out much more noisy points than BOF, thus the total query time does not increase. Further, after parallelization, the approach consumes much less time than the serial version.

Space. The dataset consists of 2072M SIFT descriptors, which occupy a space of 323G. They are impractical to load into memory and thus can not be indexed by some typical structures like E2LSH. Our index keeps only one integer for one point in each table, so each table occupies 8G space. And the parallelized version can be deployed across typical PCs without large amount of memory. If global descriptors are used, such as GISTIS[6], two orders of magnitude more images can be handled, approaching to web scale applications.

6 Conclusion

This paper proposed a new vocabulary construction algorithm for vocabulary-based hashing index. Experiment results show its efficiency which makes it more practical for large scale applications. We also described the parallelized version of the index scheme. Near-duplicate image detection experiments on a dataset with 1M images show its effectiveness and efficiency in the large scale scene.

Acknowledgments

This work was supported by the National Natural Science Foundation of China under the grant No. 60621062 and 60605003, the National Key Foundation R&D Projects under the grant No. 2007CB311003. The authors would like to thank Herve Jegou and Matthijs Douze for providing the datasets, and thank three anonymous reviewers and Jinhui Yuan for their helpful comments.

References

1. Flickr, `http://www.flickr.com`
2. Holidays dataset, `http://lear.inrialpes.fr/people/jegou/data.php`
3. Imagemagick, `http://www.imagemagick.org`
4. Arya, S., Mount, D., Netanyahu, N., Silverman, R., Wu, A.: An optimal algorithm for approximate nearest neighbor searching fixed dimensions. J. ACM (1998)
5. Datar, M., Immorlica, N., Indyk, P., Mirrokni, V.S.: Locality-sensitive hashing scheme based on p-stable distributions. In: SCG (2004)
6. Douze, M., Jégou, H., Singh, H., Amsaleg, L., Schmid, C.: Evaluation of gist descriptors for web-scale image search. In: CIVR. ACM, New York (2009)
7. Freund, Y., Schapire, R.E.: A decision-theoretic generalization of on-line learning and an application to boosting. J. of Computer and System Sciences (1997)
8. Gionis, A., Indyk, P., Motwani, R.: Similarity search in high dimensions via hashing. In: VLDB (1999)
9. Indyk, P., Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality. In: STOC (1998)
10. Jégou, H., Douze, M., Schmid, C.: Hamming embedding and weak geometric consistency for large scale image search. In: Forsyth, D., Torr, P., Zisserman, A. (eds.) ECCV 2008, Part I. LNCS, vol. 5302, pp. 304–317. Springer, Heidelberg (2008)
11. Ke, Y., Sukthankar, R., Huston, L.: Efficient near-duplicate detection and subimage retrieval. In: MM (2004)
12. Liang, Y., Li, J., Zhang, B.: Vocabulary-based hashing for image search. In: MM (to appear, 2009)
13. Lowe, D.G.: Distinctive image features from scale-invariant keypoints. In: IJCV (2004)
14. Meng, Y., Chang, E., Li, B.: Enhancing dpf for near-replica image recognition. In: Proceedings of IEEE Computer Vision and Pattern Recognition (2003)
15. Nister, D., Stewenius, H.: Scalable recognition with a vocabulary tree. In: CVPR (2006)
16. Poullot, S., Buisson, O., Crucianu, M.: Z-grid-based probabilistic retrieval for scaling up content-based copy detection. In: CIVR (2007)
17. Sivic, J., Zisserman, A.: Video Google: A text retrieval approach to object matching in videos. In: ICCV (2003)