Sequential Compact Code Learning for Unsupervised Image Hashing

Li Liu and Ling Shao, *Senior Member, IEEE*

***Abstract*— Effective hashing for large-scale image databases is a popular research area, attracting much attention in computer vision and visual information retrieval. Several recent methods attempt to learn either graph embedding or semantic coding for fast and accurate applications. In this paper, a novel unsupervised framework, termed evolutionary compact embedding (ECE), is introduced to automatically learn the task-specific binary hash codes. It can be regarded as an optimization algorithm that combines the genetic programming (GP) and a boosting trick. In our architecture, each bit of ECE is iteratively computed using a weak binary classification function, which is generated through GP evolving by jointly minimizing its empirical risk with the AdaBoost strategy on a training set. We address this as greedy optimization by embedding high-dimensional data points into a similarity-preserved Hamming space with a low dimension. We systematically evaluate ECE on two data sets, SIFT 1M and GIST 1M, showing the effectiveness and the accuracy of our method for a large-scale similarity search.**

***Index Terms*— AdaBoost, binary hash codes, genetic programming (GP), large-scale similarity search, unsupervised.**

I. Introduction

COMPACT embedding has been a critical preprocessing step in many fields of information processing and analysis, such as data mining, information retrieval [1]–[8], and pattern recognition [9], [10]. Recently, with the advances of computer technologies and the development of the World Wide Web, a huge amount of digital data, including text, images, and videos, is generated, stored, analyzed, and accessed every day. To overcome the shortcomings of text-based image retrieval, content-based image classification and retrieval have attracted substantial attention. The most basic but essential scheme for image retrieval is the nearest neighbor search: given a query image to find an image that is most similar to it within a large database and assign the same label of the nearest neighbor to this query image. However, greedily searching a data set with N samples is infeasible, because linear complexity $O(N)$ is not scalable in practical applications. Due to this kind of computational complexity problem, researchers have already developed some approaches to efficiently index data, e.g., K-D tree and R tree [11]. Nevertheless, most of these methods can only handle the data within the dimensionality of 100. In addition, most of the vision-based applications also suffer from the curse of dimensionality problems,[1] because visual descriptors usually have hundreds or even thousands of dimensions. Therefore, to make large-scale search or classification practical, some hash-based methods have been proposed to effectively reduce the dimension of data and increase the retrieval speed and accuracy.

The most well-known hashing technique that preserves similarity information is probably locality-sensitive hashing (LSH) [1]. LSH simply employs random linear projections (followed by random thresholding) to map the data points close in a Euclidean space to similar codes in the Hamming space. It is theoretically guaranteed that as the code length increases, the Hamming distance between two codes will asymptotically approach the Euclidean distance between their corresponding data points. Furthermore, kernelized LSH [2] has also been successfully proposed and utilized for large-scale image retrieval and classification. However, in realistic applications, LSH-related methods usually require long codes to achieve good precision, which result in low recall, since the collision probability that the two codes fall into the same hash bucket decreases exponentially as the code length increases.

To design effective compact hashing, a number of methods, such as projection learning for hashing, have been introduced. Salakhutdinov and Hinton [7] proposed to use stacked restricted Boltzmann machine (RBM), and showed that it is indeed able to generate compact binary codes to accelerate document retrieval. Recently, another attempt called boosted similarity sensitive coding (BSSC) [12] has also been proposed to learn a weighted Hamming embedding for a task-specific similarity search. Furthermore, principled linear projections, like PCA hashing (PCAH) [13] and its rotational variant [4], have been suggested for better quantization rather than random projection hashing. In addition, another popular technique called spectral hashing (SpH) [14] was proposed, which preserved the data locality relationship to keep the neighbors in the input space as the neighbors in the Hamming space. After that, researchers use anchor graphs to obtain tractable low-rank adjacency matrices for efficient similarity search, termed anchor graph hashing (AGH) [6]. Beyond that,

Manuscript received August 15, 2014; revised October 11, 2015; accepted October 22, 2015. Date of publication November 10, 2015; date of current version November 15, 2016. This work was supported in part by Northumbria University, in part by the National Natural Science Foundation of China under Grant 61528106, and in part by the Newton International Exchanges Scheme. *(Corresponding author: Ling Shao.)*

The authors are with the Department of Computer Science and Digital Technologies, Northumbria University, Newcastle upon Tyne NE1 8ST, U.K. (e-mail: li2.liu@northumbria.ac.uk; ling.shao@ieee.org).

[1]The effectiveness and efficiency of these methods drop exponentially as the dimensionality increases, which is commonly referred to as the curse of dimensionality.

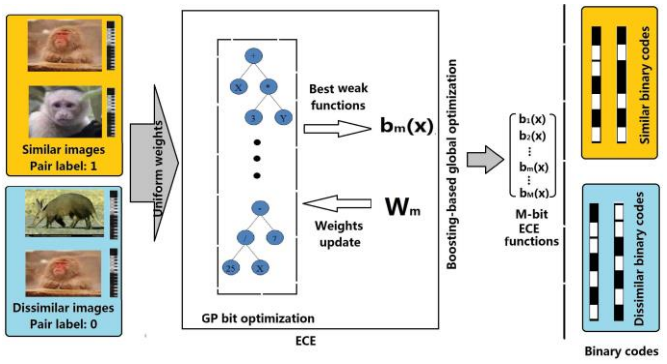


Fig. 1. Workflow of ECE. We learn ECE on these training data with GP bit optimization and boosting-based global optimization, and finally obtain the optimized embedding function, which can be directly used to embed the feature from a high-dimensional space into a lower binary one.

self-taught hashing (STH) [15], latent structure preserving hashing [16], spherical hashing (SpherH) [5], iterative quantization (ITQ) [4], compressed hashing (CH) [17], and so on have also been effectively applied for large-scale data retrieval tasks.

Although the existing embedding methods achieve promising results in a variety of applications, they basically rely on complex and advanced mathematical knowledge to optimize the predefined objective functions. However, for some optimization problems, direct solutions cannot always be found. Besides, in large-scale settings, matrix factorization techniques used in the above methods can also cause a heavy computational burden. Therefore, how to automatically generate better solutions to optimization problems becomes an interesting topic for real-world vision applications. In this paper, we propose a novel method, termed evolutionary compact embedding (ECE), which applies genetic programming (GP) in combination with AdaBoost to automatically solve accurate and robust large-scale retrieval problems. A key advantage of using GP is that the hash functions computed by these weak classifiers are not fixed but evolved, unlike existing embedding methods. Fig. 1 shows the workflow of ECE.

GP simulates the Darwinian principle of natural selection to solve optimization problems [18]. Different from other handcrafted techniques based on deep domain knowledge, GP is inspired by natural evolution and can be employed to automatically solve problems without prior knowledge of the solutions. Users can use GP to solve a wide range of practical problems, producing human-competitive results and even patentable inventions. Relying on natural and random processes, GP can escape traps by which deterministic methods may be captured. Because of this, usage of GP is not limited to any research domain, and creates relatively generalized solutions for any target tasks. A population in GP is allowed to evolve (using crossover and mutation) through sexual reproduction with single or pair parents chosen stochastically while biased in their fitness on the task at hand. In this way, the general fitness of population tends to improve over time. Finally, the obtained individual that achieves the best performance is taken as the final solution. More details of GP can be found in [18]–[21].

Aiming for the task of data retrieval, we intentionally combine GP (learning the weak functions) with a boosting trick to obtain a novel embedding method. For an *M*-bit embedding, GP is used to iteratively generate a best performing weighted binary classifier for each bit by jointly minimizing its empirical risk with the gentle AdaBoost strategy [22] on a training set. This embedding scheme reduces the Hamming distance between the data from the same class, while increasing the Hamming distance for data from different classes. The final optimized representation is defined as the code calculated from the nonlinear GP-evolved binary learner for each embedding bit.

The remainder of this paper is organized as follows. A brief review of related work is given in Section II. In Sections III and IV, the architecture of ECE and the implementation details are presented. Experiments and results are described in Section V. In Section VI, we conclude this paper and outline the possible future work.

II. Related Work

Recently, some techniques have been successfully used for feature embedding based on boosting schemes. One of the most related works is called BSSC [12], which is designed to learn an *M*-bit weighted Hamming embedding for a task-specific similarity search as follows:

$$H : X \rightarrow \{\alpha_1 h_1(x), \ldots, \alpha_m h_m(x), \ldots, \alpha_M h_M(x)\} \quad (1)$$

so that the distance between any two samples x_i and x_j is given by a weighted Hamming distance

$$D(x_i, x_j) = \sum_{m=1}^{M} \alpha_m |h_m(x_i) - h_m(x_j)| \quad (2)$$

where the weight α_m and the function $h_m(x_i)$ are the binary regression stumps that map the input vector x_i into binary features and are learned using boosting.

In their implementation, the training data are pairs of similar or dissimilar samples, and the weak classifiers are thresholded projections that assign a positive or a negative label to a pair. The true label of a pair (x_i, x_j) corresponds to the underlying similarity $S(x_i, x_j)$.

By applying the architecture of BSSC, a related work for fast vision applications has been carried out by Shakhnarovich *et al.* [23], in which each image is represented by a binary vector calculated via boosting coding. For the learning stage, positive examples are pairs of images x_i, x_j, so that x_j is one of the nearest neighbors of x_i, $j \in NN(x_i)$. Negative examples are pairs of images that are not neighbors. In their work, Gentle AdaBoost is used with regression stumps to minimize the exponential loss. The corresponding details can also be seen in [24].

Trzcinski *et al.* [25] proposed a descriptor called low-dimensional boosted gradient map (L-BGM), whose similarity measure models the correlation between weak learners leading to a compact description. They optimized over gradient-based features resulting in a learned representation that closely resembles the well-known SIFT. Although highly accurate, L-BGM computes a floating point descriptor, and therefore, its matching time is costly.

Then, an improved work related to [25] is presented in [26]. The boosting trick is employed to learn discriminative binary descriptors for image classification under illumination and viewpoint changes. Leveraging the boosting trick, they simultaneously optimize both the descriptor weighting and the pooling strategies. The proposed sequential learning scheme finds a single boosted hash function per dimension as a linear combination of nonlinear gradient-based weak learners. The binary hash function relies on weak learners that are applied directly to the image patches, which frees the method from any intermediate representation and allows it to automatically learn the image gradient pooling configuration of the final descriptor. Inspired by the success of the above works, in this paper, we aim to combine the boosting trick with GP to learn the compact binary codes for large-scale information retrieval tasks. A similar work has been done with supervised ECE on large-scale classification problems in [27].

III. Evolutionary Compact Embedding

In this section, the overall design of our evolutionary embedding algorithm is first introduced, and then, we describe how to train our GP classifier with the boosting trick.

A. Problem Formulation

Let us now consider the M-bit ECE Code $= [b_1(x), \ldots, b_m(x), \ldots, b_M(x)]$, which maps the high-dimensional representation into an M-dimensional string. Here, $b_m(x) \in \{1, 0\}$ is defined by $b_m(x) = \text{binary}(f_{\text{gp}}(x))$, where $f_{\text{gp}}(x)$ indicates the classifier generated by GP and the binary() function returns 1 if the argument is positive, and 0 otherwise. For clearer illustration, here is an intuitive example: if $f_{\text{gp}}(x) = 1.2$, $\text{binary}(f_{\text{gp}}(x)) = 1$, while if $f_{\text{gp}}(x) = -0.8$, $\text{binary}(f_{\text{gp}}(x)) = 0$.

Since our $f_{\text{gp}}(x)$ is originally designed for binary classification problems, here we use the pairwise trick to transfer the multiclass classification issue to a binary one. Given a set of training samples $X = \{x_1, x_2, \ldots, x_n, \ldots, x_N\}$ with labels $Y = \{1, 2, \ldots, C\}$, we redistribute them into a pairwise format $X_{\text{pair}} = \{\ldots (x_n, x_p)_j, \ldots\}_{j=1}^{N^2}$ with labels $Y_{\text{pair}} = \{1, 0\}$. X_{pair} is the set of N^2 labeled training pairs, such that $Y_{\text{pair}} = 1$ if pair data x_n and x_p belong to the same class, and $Y_{\text{pair}} = 0$ otherwise.

However, for realistic scenarios, we cannot get the precise label for each of the data points in large-scale retrieval tasks. Thus, we use an approximate scheme to obtain the weak label information. In particular, we first adopt a clustering method (e.g., K-means) to partition the data into several groups. Since this kind of clustering method is normally based on distances (e.g., Euclidean distance) to divide data into different groups, data points from the same cluster always have high similarity. Therefore, we assign pair label $Y_{\text{pair}} = 1$ if pair data x_n and x_p belong to the same cluster (group), and $Y_{\text{pair}} = 0$ if pair data x_n and x_p come from different clusters (groups).

In our approach, any two samples in X should be assigned together once to form a data pair, and we will need $N \times N = N^2$ pairs in total to obtain all the possible pairs.

Now, we can involve binary classifiers into iterative AdaBoost learning to jointly minimize its empirical loss

$$L(X_{\text{pair}}, Y_{\text{pair}}, M) = \sum_{j=1}^{N^2} \text{binary}\left(\sum_{m=1}^{M} D_m \cdot F_{\text{weak}}^m(j)\right) \div = Y_{\text{pair}}(j) \quad (3)$$

where $F^m_{\text{weak}} = b_m(x_n) \quad b_m(x_p)$ calculates the result of data pair $(x_n, x_p)_j$ using the mth weak classifier. In particular, F^m_{weak} returns 0, when $b_m(x_p)$ and $b_m(x_q)$ are different, and returns 1 otherwise. D_m is the mth coefficient corresponding to $F^m_{\text{weak}} D_m$ controls and adjusts the pair-data classification result for each bit. The similar pairwise formulation can also be seen in [10] and [28]–[31].

Equation (3) reflects the final error rate on the classification of X_{pair} using the ensemble of weak classifiers. Minimizing (3) aims at reducing the Hamming distances between high-dimensional data from positive pairs ($Y_{\text{pair}} = 1$) while increasing the Hamming distances between high-dimensional data from negative pairs ($Y_{\text{pair}} = 0$). The optimization problem in Equation (3) seems to be related to the standard AdaBoost formulation. However, the F_{weak} functions are much more complex than the one used in standard AdaBoost, since F_{weak} is a product of two classifiers, i.e., $F^m_{\text{weak}} = b_m(x_n) \quad b_m(x_p)$. The current optimization of (3) is intrinsically discontinuous and highly nonconvex, and in practice, the space of all possible weak learners b_m is discrete and prohibitively large. To better tackle (3), in this paper, we use a greedy optimization algorithm, i.e., GP, to automatically create binary classifiers for this optimization problem.

In particular, for each bit, we evolve the entire GP system once to generate a relatively effective weak classifier F^m_{weak} (i.e., $\text{Error}_{\text{rate}} < 0.5$) under weighted data distribution. By adopting the boosting scheme, ECE is iteratively optimized over the same-labeled and differently labeled sample pairs. Initially, each data pair is assigned the same weight value.[2] At each iteration, incorrectly embedded samples, i.e., the pairs of differently labeled samples mistakenly regarded as from the same labels, are assigned larger weights, while the weights of correctly embedded samples are reduced. Hence, the next bits tend to correct the errors of the preceding ones.

ECE computes each bit for samples through the GP bit optimization procedure. Based on the result (i.e., error rate) calculated from each bit, the boosting scheme is then applied as a global optimization to balance the weights of different GP classifiers. Thus, the final loss function (3) will be decreased effectively using this kind of weighted ensemble of GP classifiers. In Section III-B, we describe our GP bit optimization and boosting-based global optimization algorithms.

B. Genetic Programming Bit Optimization

GP is an evolutionary computation (EC) technique that automatically solves problems without requiring the user to know or specify the form or structure of the solution in advance.

[2]The weight value is later defined as w in (4). To avoid confusion with D, all the weight values mentioned in this paper indicates w, while D denotes the coefficient corresponding to F_{weak}.

TABLE I
FUNCTION SET IN GP

Function Name	Input	Description
$+$	x_1 and x_2	$y = x_1 + x_2$
$-$	x_1 and x_2	$y = x_1 - x_2$
$\times$	x_1 and x_2	$y = x_1 \times x_2$
$\div$	x_1 and x_2	$y = \frac{x_1}{\sqrt{1+x_2^2}}$
IF	x_1, x_2 and x_3	if $x_1 < 0$, $y = x_2$; otherwise $y = x_3$

In general, GP programs can be represented as a tree structure during the evolution procedure. In this paper, each individual in GP represents a candidate binary classifier and is evolved continuously through generations. To establish the architecture of our model, three important concepts, such as function set, terminal set, and fitness function, should be defined.

1) Terminal Set and Function Set: Individuals in the population are assembled from terminal and function nodes. Terminal nodes are used as the input to the genetic program and are taken from the terminal set. We used two kinds of terminals in the terminal set:

1) feature terminals corresponding to the image features;
2) constant terminals that are the randomly generated constant numbers.

Similar to other example-based learning algorithms, these terminals remain unchanged throughout the learning process. In our classification model, we define pair data X_{pair} and random constant numbers between 0 and 1 as the terminal set for GP evolving. In each tree-based genetic structure, data are located at the bottom leaf of the entire tree and connect with the higher function nodes directly.

In addition, another key component of GP is the function set that constitutes the internal nodes of the tree and is typically driven by the nature of the problem. Usually, for GP classification problems, "+," "−," "×," and "÷" are adopted in the function set. The "+," "−," and "×" operators are used as their original meanings, i.e., addition, subtraction, and multiplication. However, "÷" is different from general division or protected division. In our model, "÷" is called analytic division, which is proved leading to better results in GP regression problems [32]. Each of these four operators takes two arguments and returns one result. In addition, we get another conditional function "if" with three arguments. If the first is negative, the second argument is returned; otherwise, it returns the third argument. The "if" function allows a program to contain a different expression in different regions of a feature space, and allows discontinuous programs rather than insisting on smooth functions [33]. Table I lists all these functions used in our GP classification model.

2) GP Classification Scheme: Each GP classifier is represented as a tree-based classifier and returns a real value as output. In this way, there is a problem in this method of classification. This is because the task of binary classification requires a binary output rather than a continuous range of values as returned by the numeric expression representation. Therefore, a process of interpretation must be applied to convert the numeric output into a binary one. For two-class problems, the division point between the negative and nonnegative numbers forms a natural boundary between the classes. Thus, in our model, we set zero as a boundary to separate two classes. If the GP output is positive, the example is predicted as belonging to one class, and the other class otherwise. Fig. 2 illustrates how we use the output of a genetic program for binary classification. Numeric expressions have a hierarchical tree structure, which naturally suits the GP architecture. For numeric expressions to be evolved by the GP evolutionary search algorithm, a fitness measure must be derived.

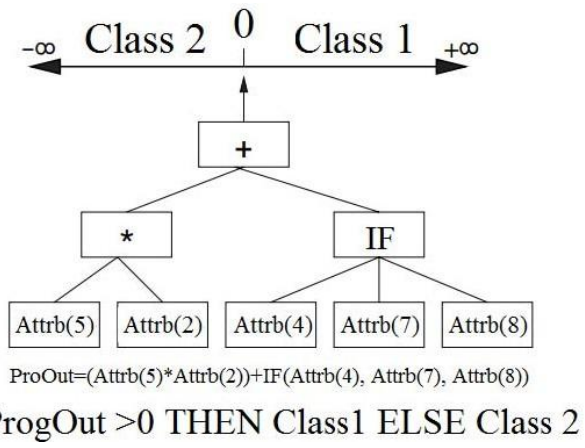


Fig. 2. Classification strategy using a GP program. *Attrb(i)* indicates the *i*th value of the input vectors.

The reason why GP classifiers are used in our method is mainly driven by the loss function in (3). In particular, most of machine learning classification algorithms always need predefined formulations to optimize, which are fixed and based on deep domain knowledge. For instance, the objective formulation of SVM is always fixed, i.e., $\min||w||^2$ s.t. $y_i(w^{\mathsf{T}}x_i + b) \geq 1$, and its solution needs specific and complicated mathematical derivation. In our task, such classifiers fail to directly find the analytical solutions to optimize the objective functions in (3), since their fixed architecture of formulations is not suitable due to the intrinsic structure in (3). However, GP is flexible and is not based on any fixed formula or structure. Moreover, it can allow the computer to automatically solve tasks without requiring users to know or specify the form or structure of the solution in advance according to [18]. Thus, it is intuitive for solving the problems and easy to implement our task through GP. Furthermore, the optimization of (3) is discontinuous and highly nonconvex, and in practice, the space of all possible weak learners b_m is discrete and prohibitively large. GP is regarded as a nondeterministic algorithm that can achieve a flexible search space and effectively solve highly nonlinear optimization problems compared with other fixed structured classifiers. Besides, a GP classifier is a tree-based classifier, which is indeed relatively simple but can still lead to good results. We have also stated that our GP classifier is better than other classifiers (including the normal decision tree via

the C4.5 algorithm) [34], [35]. Thus, motivated by the above reasons, we develop a greedy optimization algorithm, i.e., GP, for solving this difficult problem.

3) Fitness Function: The fitness function in GP determines how well a program is able to solve the problem. For separating the pairwise samples (e.g., x_n and x_p) into positive (pairs of samples from the same class) or negative (pairs of samples from different classes), we use $F^m_{\text{weak}} \equiv b_m(x_n) \quad b_m(x_p)$ to distinguish each pair of data. $b_m(x)$ is the GP classifier for the mth bit. Assuming an N^2 pairwise sample data set X_{pair} and their labels $Y_{\text{pair}} \in \{1, 0\}$, we run the GP system for each bit, and the corresponding fitness function for the mth bit is designed as follows:

$$\text{fitness}^m = \left\lfloor \sum_{j=1}^{N^2} \delta_j^m w_j^m \right\rfloor \times 100\% \tag{4}$$

where δ_j^m is equal to 1 if $F^m_{\text{weak}}(j) \neq Y_{\text{pair}}(j)$ and 0 otherwise, $F^m_{\text{weak}}(j)$ indicates the output of the j th pair samples, $Y_{\text{pair}}(j)$ denotes the label of the j th pair samples, and w_j^m is the weight of the j th pair samples for the mth bit. This fitness function calculates the error rate by summing the weights of those wrongly classified data pairs. This is very similar to the AdaBoost by measuring the goodness of a weak hypothesis. In this way, GP can effectively get a relatively precise binary classification by continuously minimizing the value of fitness during the whole evolution procedure.

For large-scale data sets, the fitness function must be evaluated many times in each GP generation. For getting good results, a large number of generations are usually required, which lead to heavy computation. In our experiments, we implement parallel processing to speed up the GP learning algorithm. In our implementation, the large number of fitness evaluation can be performed by multiple processors at the same time, giving a tremendous reduction in the training time.

4) Evolutionary Parameters: For GP evolution, a lexicographic parsimony pressure has been applied as the selection method in our running. Like the original selection method, a random number of individuals are chosen from the population, and then, the best of them is selected. The only difference from the original selection is that, if multiple individuals are equally fit, the shortest one (the tree with the least number of nodes) is chosen as the best. Lexicographic parsimony pressure has shown its effectiveness for controlling the bloat [18] in different types of problems. In addition, we have adopted the totalelitism scheme as the survival module, in which all the individuals from both parents and children populations are ordered by fitness alone, regardless of being parents or children. This scheme has been demonstrated to lead to promising results in many applications. The ramped half-and-half method [36] was used for generating programs in the initial population. Table II shows these relevant parameters for GP evolving. In our implementation, since each GP classifier is evolved as a weak learner for the AdaBoost architecture, we empirically set the maximum number of generations as 50, which is proved to be enough for obtaining an acceptable weak learner (i.e., yielding a classification error lower than 50%) in this case.

TABLE II
PARAMETERS FOR OUR GP ALGORITHM

GP parameters	Values
Population size	300
Generation size	50
Crossover rate	75%
Mutation rate	20%
Elitism rate	5%
Selection for reproduction	'lexictour'
Survival method	'totalelitism'
Stop condition	$\leq 0.1\%$

Population size indicates the number of individual computer programs, which are composed of the available functions and terminals, in the initialization phase. *Generation size* indicates the number of iterations during the whole evolving optimization. *Crossover rate* denotes the percentage of creating new offspring program(s) for the new population by recombining randomly chosen parts from two selected programs. Mutation rate denotes the percentage of creating one new offspring program for the new population by randomly mutating a randomly chosen part of one selected program. *Elitism rate* denotes the percentage of directly copying the selected individual program to the new population. The value of the *stop condition* is regarded as the lower bound of the fitness function's cost for stopping the whole evolving procedure.

C. Boosting-Based Global Optimization

In Sub-Section III-B, we presented the theoretical algorithm for calculating each bit for the ECE code. However, we still have not mentioned how to get the coefficient D_m for minimizing the loss function [see (3)]. To make our optimization convenient, we directly follow the gentle AdaBoost scheme to update D_m for each bit of ECE. Gentle AdaBoost is a more robust and stable version of the real AdaBoost (see [22] for a full description). So far, it has been the most practically efficient boosting algorithm used, for example, in the Viola–Jones face detector [37]. Previous experiments show that gentle AdaBoost performs slightly better than real AdaBoost on regular data, but is considerably better on noisy data and much more resistant to outliers.

In our model, F^m_{weak} with the lowest error rate $E_r = \left[\sum_{F^m_{\text{weak}}(j) \neq Y_{\text{pair}}(j)} w_j^m\right]$ by 4 is selected as the best solution for the current mth bit after GP evolving. The corresponding coefficient D_m for this F^m_{weak} can be then represented as $D_m = 1 - 2E_r$. For the next-bit GP optimization, w_j^{m+1} for the j th training sample pair can be updated as

$$w_j^{m+1} = \frac{w_j^m \exp\left[-D_m Y_{\text{pair}}(j) F^m_{\text{weak}}(j)\right]}{\sum_{j=1}^{N^2} w_j^m \exp\left[-D_m Y_{\text{pair}}(j) F^m_{\text{weak}}(j)\right]}. \tag{5}$$

Note that, for the first bit ($m = 1$) of GP optimization, each data pair in the N^2 samples training set is initialized as the equal weight: $w_j^{m=1} = (1/N^2)$.

According to the above boosting-based global optimization, we can summarize the mechanism of our ECE algorithm as follows: given the existing training pairs X_{pair} and their corresponding labels Y_{pair}, ECE can learn a boosted hash function $b_m(x)$ for each binary bit. In particular, each $b_m(x)$ is iteratively optimized over similar and dissimilar sample pairs of data in an individual GP optimization procedure with an updated sample weight w in the fitness function

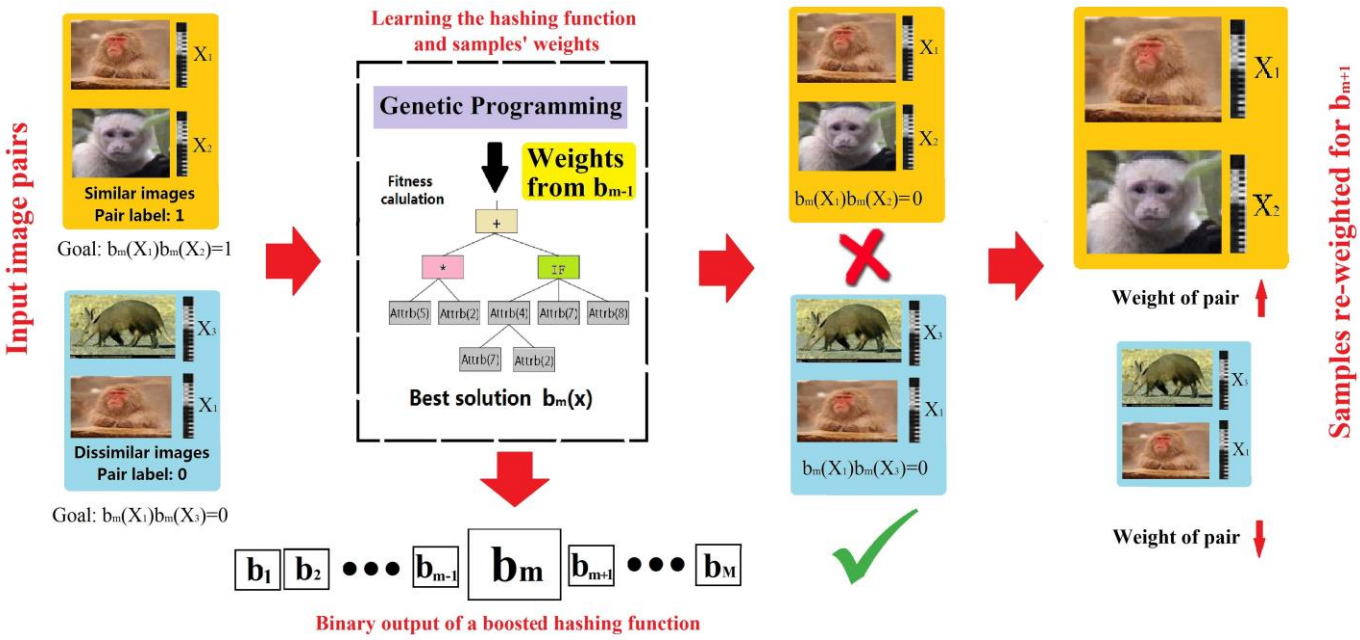


Fig. 3. Each bit of ECE is iteratively optimized by GP over the same-labeled (i.e., pair label:1) and differently labeled (i.e., pair label:0) sample pairs. Initially, each data pair is assigned the same weight. At each iteration, incorrectly embedded samples, such as the pairs of same-labeled samples mistakenly assigned to the different embedding values (e.g., $b_m(x_1)b_m(x_2) = 0$), are assigned a larger weight, while the weight of correctly embedded samples is reduced. Hence, the next bit tends to correct the errors of the preceding ones.

(the reason why using GP in our method is explained in Sub-Section III-B). At each iteration, incorrectly hashed samples, e.g., a pair of data from the same cluster (clusters are generated by K-means) mistakenly assigned different values by $b_m(x)$, are given a larger weight, while the weight of the correctly hashed pair of samples is reduced. Hence, the next bit tends to correct for the errors of the preceding ones to jointly minimize its empirical risk with the AdaBoost strategy. This embedding scheme effectively reduces the Hamming distance between the data from the same cluster (i.e., similar data), while increasing the Hamming distance for data from different clusters (i.e., dissimilar data). This scheme using the AdaBoost strategy is more powerful for hashing than those used in most of the previous work on binary embedding, since the AdaBoost strategy will make our hashing code more compact and discriminative. In this way, to compute an M-bit ECE code, we need to repetitively run GP M times. After ECE learning on the training set, for a new high-dimensional data x, the final ECE code is represented as Code $= [b_1(x), \ldots, b_m(x), \ldots, b_M(x)]$. Fig. 3 visualizes the procedure of the ECE optimization scheme.

It is noteworthy that our approach can be regarded as an embedding learning method. Once our ECE embedding functions are obtained by GP, they are fixed and then can be directly used on any new coming data similar as a handcrafted embedding scheme without relearning. The corresponding algorithm is depicted in Algorithm 1.

IV. Improved ECE Implementation for Large-Scale Applications

Our ECE method can theoretically reduce data of any dimension to a lower dimension compact code. However, the GP algorithm is always time-consuming for training on large-scale data sets, especially when the dimensionality of the original data is high.

To reduce the GP optimization complexity, we improve our ECE algorithm using the random batch parallel learning (RBPL) technique. Given a training set $X = \{x_1, x_2, \ldots, x_n, \ldots, x_N\}$ with labels $Y = \{1, 2, \ldots, C\}$, we randomly assemble them into N pairs $\hat{X}_{\text{pair}} = \{\ldots (x_n, x_p)_j, \ldots\}_{j=1}^{N}$ with labels $\hat{Y}_{\text{pair}} = \{1, 0\}$ using the half–half scheme,[3] instead of generating a full-possibility N^2-sized data pair set X_{pair}. We repeat this kind of random assignment K times, so that K groups of pair data sets are obtained: $\{\hat{X}^1_{\text{pair}}, \hat{X}^2_{\text{pair}}, \ldots, \hat{X}^K_{\text{pair}}\}$ with their corresponding pair labels $\{\hat{Y}^1_{\text{pair}}, \hat{Y}^2_{\text{pair}}, \ldots, \hat{Y}^K_{\text{pair}}\}$.

In this way, we can use parallel computation to separately learn an M-bit ECE for each $\hat{X}_{\text{pair}}$ at same time. We further concatenate these ECE codes into a long code. Although using the original full-possibility training set, X_{pair} can theoretically learn a better ECE code than just applying any single N-pair set $\hat{X}_{\text{pair}}$; in fact, the ECE codes calculated in parallel from different randomly assigned sets $\hat{X}_{\text{pair}}$ can effectively compensate each other's training errors (better resisting over-fitting). Therefore, the concatenated code can still keep the smallest Hamming distance for data from the same class and enlarge the Hamming distance for data from different classes. In terms of complexity, if each bit GP optimization needs a population of S individuals evolved by T generations,

[3]The half–half scheme aims to balance the training data by generating half of the pairwise data belonging to label 1 and the rest of pairs belonging to label 0. This scheme makes the training samples evenly fill the data space and effectively reduce the overfitting in the training phase.

Algorithm 1 Evolutionary Compact Embedding

Input: A training set containing N^2 pairs of data $X_{pair} = \{\ldots (x_n, x_p)_j, \ldots\}_{j=1}^{N^2}$;

Aim: Learn an M-bits embedding code

First step

(1) Assign labels $Y_{pair} \in \{1, 0\}$ with K-means, where $Y_{pair} = 1$ if pair data x_n and x_p belong to the same cluster, and $Y_{pair} = 0$ otherwise;

(2) Initialize data weights: $w_j^{m=1} = \frac{1}{N^2}$ for the first bit optimization;

Second step

For $m = 1, \ldots, M$:

1. Complete the GP bit optimization procedure to obtain best-performing F_{weak}^m with the fitness function Equation 4, where $F_{weak}^m(j) = b_m(x_n) \quad b_m(x_p)$;

2. For each pair of data, the evolved weak classifier F_{weak}^m calculates: $X_{pair} \rightarrow \{1, 0\}$. The error rate is evaluated with respect to $E_r = [\sum_{1}^{N^2} \delta_j^m w_j^m]$;

3. If $E_r >= 0.5$, **STOP** loop; Otherwise, **CONTINUE**;

4. Calculate the coefficient D_m of this F_{weak}^m: $D_m = 1 - 2E_r$;

5. Update the weights of the N^2 pairs of training data: $w_j^{m+1} = \frac{w_j^m \exp(-D_m Y_{pair}(j) F_{weak}^m(j))}{\sum_{j=1}^{N^2} w_j^m \exp(-D_m Y_{pair}(j) F_{weak}^m(j))}$;

End

Output:

The M-bits ECE code expression: $Code = [b_1(x), \ldots, b_m(x), \ldots, b_M(x)]$, where x is a new high-dimensional feature.

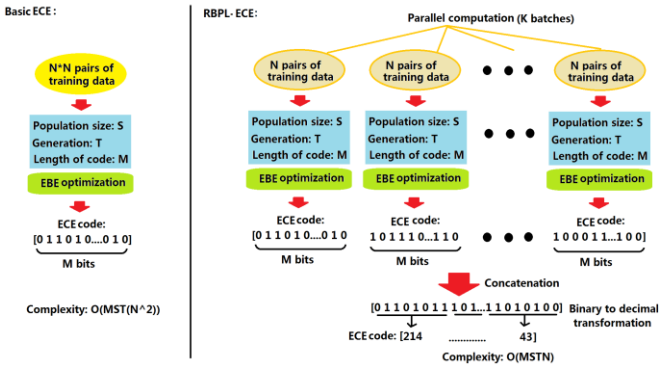


Fig. 4. Comparison between basic ECE and RBPL-ECE.

for embedding M bits using the basic ECE algorithm, the training complexity is $O(\text{MSTN}^2)$. Our RBPL technique can effectively reduce the basic ECE training complexity from $O(\text{MSTN}^2)$ to $O(\text{MSTN})$. Thus, the RBPL-ECE implementation is about N times faster than the basic ECE algorithm. Fig. 4 illustrates our RBPL-ECE implementation.

V. Experiments and Results

In this section, ECE algorithm has been evaluated on the high-dimensional nearest neighbor search problem. Following the previous reports, two large-scale realistic data sets[4] are used in our experiments, i.e., SIFT 1M and GIST 1M, which both contain one million image features with 128 −dim and 960 − dim vectors, respectively.

For each data set, we randomly select 10k data points as queries and use the remaining to form the gallery database for training. We generate the ground truth using the same criterion as in [38]. In the test phase, similarly, a returned point is regarded as a true neighbor if it lies in the top two percentile points closest to a query. Hamming distances ranking is then used as the measurement for in our retrieval tasks, since it is fast enough with short hash codes in practice. We evaluate the retrieval results by the mean average precision (MAP) and the precision–recall curve. In addition, we also report the training time and the testing time (the average time used for each query) for all the methods. Our experiments are completed using MATLAB 2013a on a server configured with a 12-core processor and 128G of RAM running the Linux OS.

A. Compared Methods and Settings

We compare our method against 13 popular hashing algorithms, i.e., locality-sensitive hashing (LSH) [1], kernelized locality-sensitive hashing (KLSH) [2], RBM [39], BSSC [12], PCAH [13], SpH [14], AGH [6], STH [15], KSH [40], SpherH [5], ITQ [4], LLH [41], and CH [17]. In particular, for KLSH and KSH, we both use the RBF kernel and randomly sample 500 training samples to construct the empirical kernel map and set the scalar parameter σ always to an appropriate value on each data set. To run RBM, we train it with a set of 100 − 100 hidden layers without fine-tuning. BSSC uses the labeled pair scheme mentioned above in a boosting framework to learn the thresholds and weights for each hash function. AGH with two layers is used in our comparison, which shows superior performance over AGH with one layer [6]. We further set $k = 200$ as the number of the anchor points and the number of nearest anchors in sparse coding as $s = 50$. Both our CH method and the AGH need an anchor-based sparse coding step, and thus, the same settings are also applied in CH. The settings for other methods have also strictly followed the original reports. For our method RBPL-ECE, since we mainly evaluate the short hash codes, we just fix the batch number as $K = 4$ in all experiments. The number of clusters of K-means in the proposed method for each data set is selected from one of {600, 700, 800,..., 1000,... , 1500} with the step of 100, which yields the best performance by tenfold cross-validation. Due to that basic ECE and RBPL-ECE are both inspired by GP, which is always initialized randomly, all the experiments with our methods have been repetitively car- ried out ten times and the final results shown are the averages of the ten runs with a degree of uncertainty. All of the above methods in our experiments are evaluated on six different lengths of codes (16, 32, 48, 64, 80, and 96).

B. Results Comparison

Fig. 5 illustrates the MAP curves of all comparable algorithms on SIFT 1M and GIST 1M data sets. In its entirety,

[4]Download here: http://corpus-texmex.irisa.fr/

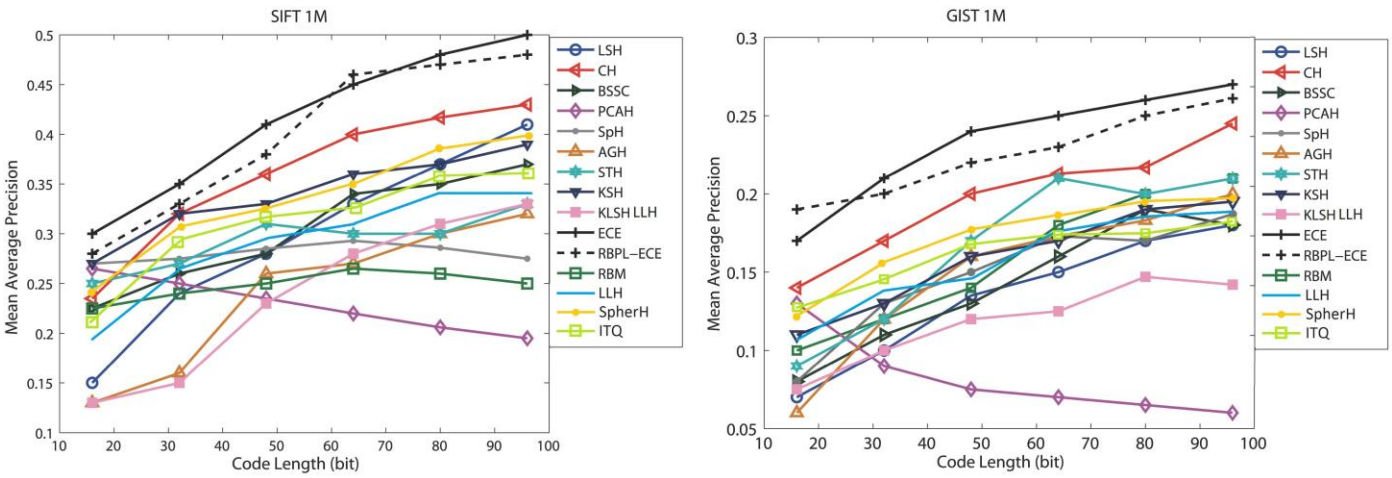


Fig. 5. MAP of all the algorithms on SIFT 1M and GIST 1M data sets.

TABLE III

MAP OF 32 AND 48 b WITH TRAINING AND TESTING TIME OF ALL ALGORITHMS ON SIFT 1M AND GIST 1M DATA SETS

Methods	SIFT 1M						GIST 1M					
	32 bits			48 bits			32 bits			48 bits		
	MAP	Train time	Test Time	MAP	Train time	Test Time	MAP	Train time	Test Time	MAP	Train time	Test Time
LSH	0.240	**0.3**s	**1.1**μs	0.280	**0.6**s	**1.9**μs	0.107	**1.4**s	**2.7**μs	0.135	**2.1**s	**3.0**μs
KLSH	0.150	10.5s	14.6μs	0.230	10.7s	16.2μs	0.110	29.5s	27.2μs	0.120	30.7s	38.0μs
RBM	0.245	4.5×10^4s	3.3μs	0.262	5.8×10^4s	3.7μs	0.123	5.5×10^4s	3.4μs	0.142	6.2×10^4s	3.7μs
BSSC	0.280	2.2×10^3s	11.2μs	0.293	2.6×10^3s	13.4μs	0.112	3.2×10^3s	13.0μs	0.130	3.8×10^3s	15.1μs
PCAH	0.252	6.5s	1.2μs	0.235	7.4s	**1.9**μs	0.090	49.2s	2.8μs	0.075	52.3s	**3.0**μs
SpH	0.275	25.8s	28.3μs	0.284	88.2s	101.9μs	0.130	65.3s	40.2μs	0.148	131.1s	116.3μs
AGH	0.161	144.7s	55.7μs	0.267	184.2s	72.0μs	0.134	242.5s	83.7μs	0.160	279.4s	95.6μs
STH	0.270	1.2×10^3s	17.4μs	0.318	1.8×10^3s	19.8μs	0.123	1.9×10^3s	21.3μs	0.171	2.5×10^3s	25.2μ
KSH	0.324	2.1×10^3s	32.0μs	0.339	2.3×10^3s	37.4μs	0.136	2.6×10^3s	34.2μs	0.161	2.9×10^3s	38.1μs
CH	0.320	93.4s	53.5μs	0.360	98.2s	54.4μ	0.175	194s	64.1μs	0.206	210.5s	71.5μs
ITQ	0.275	721.0s	32.1μs	0.319	896.3s	35.7μ	0.144	1.34×10^3s	33.8μs	0.167	1.57×10^3s	36.0μs
SpherH	0.303	1.03×10^3s	39.4μs	0.330	1.12×10^3s	38.1μ	0.158	1.92×10^3s	36.2μs	0.175	2.33×10^3s	36.4μs
LLH	0.259	988.4s	25.0μs	0.287	1.24×10^3s	28.7μ	0.138	1.57×10^3s	27.4μs	0.143	1.90×10^3s	29.8μs
(Basic) ECE	**0.350** $\pm$**0.009**	5.5×10^4s	28.2μs	**0.412** $\pm$**0.014**	6.2×10^4s	31.7μs	**0.233** $\pm$**0.018**	6.1×10^4s	30.8μs	**0.249** $\pm$**0.017**	6.7×10^4s	35.0μs
RBPL-ECE	**0.335** $\pm$**0.012**	917s	32.1μs	**0.387** $\pm$**0.020**	984s	33.5μs	**0.204** $\pm$**0.024**	1.7×10^3s	32.7μs	**0.213** $\pm$**0.021**	1.9×10^3s	33.9μs

The best/fastest results in this table are bold. The results of EBE and RBPL-EBE are mean accuracies of 10 runs with a degree of uncertainty.

the searching accuracies on the SIFT 1M data set are obviously higher than that on the more complicated GIST 1M data set. In particular, for the PCAH, it has a high MAP when the code length is short. However, it fails to make significant improvements, and the performance decreases as the code length increases. On the contrary, the rest of the methods keep an overall increasing or fluctuating tendency on MAP when the code length increases. In particular, LSH and KLSH have a low MAP when the code length is short. STH and BSSC always produce competitive search accuracies on both the data sets. The performance of the RBM and SpH achieves rise-then-fall curves on the SIFT 1M data set. Beyond those, it is obviously observed that KSH and CH always reach high search accuracies on both the data sets. For our methods, both ECE and RBPL-ECE can significantly outperform the other comparable methods (also shown in Table III) in terms of the MAP. Fig. 6 also presents the precision–recall curves of all the algorithms on two data sets with the code of 48 b. From both the figures in Fig. 6, we can further discover that, for both the data sets, the (basic) ECE achieves slightly better

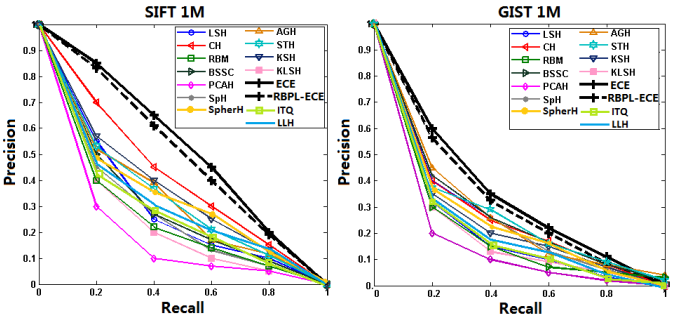


Fig. 6. Precision–recall curves of all algorithms on SIFT 1M and GIST 1M data sets for the codes of 48 b.

performance than RBPL-ECE by comparing the MAP and area under the curve. The learned GP-tree-based hashing functions for embedding 16-b binary codes on the SIFT 1M data set are illustrated in Fig. 7. In addition, we also give a numerical example of the fifth bit hash function in Fig. 7 to show how to compute a bit using the learned GP-tree-based hash

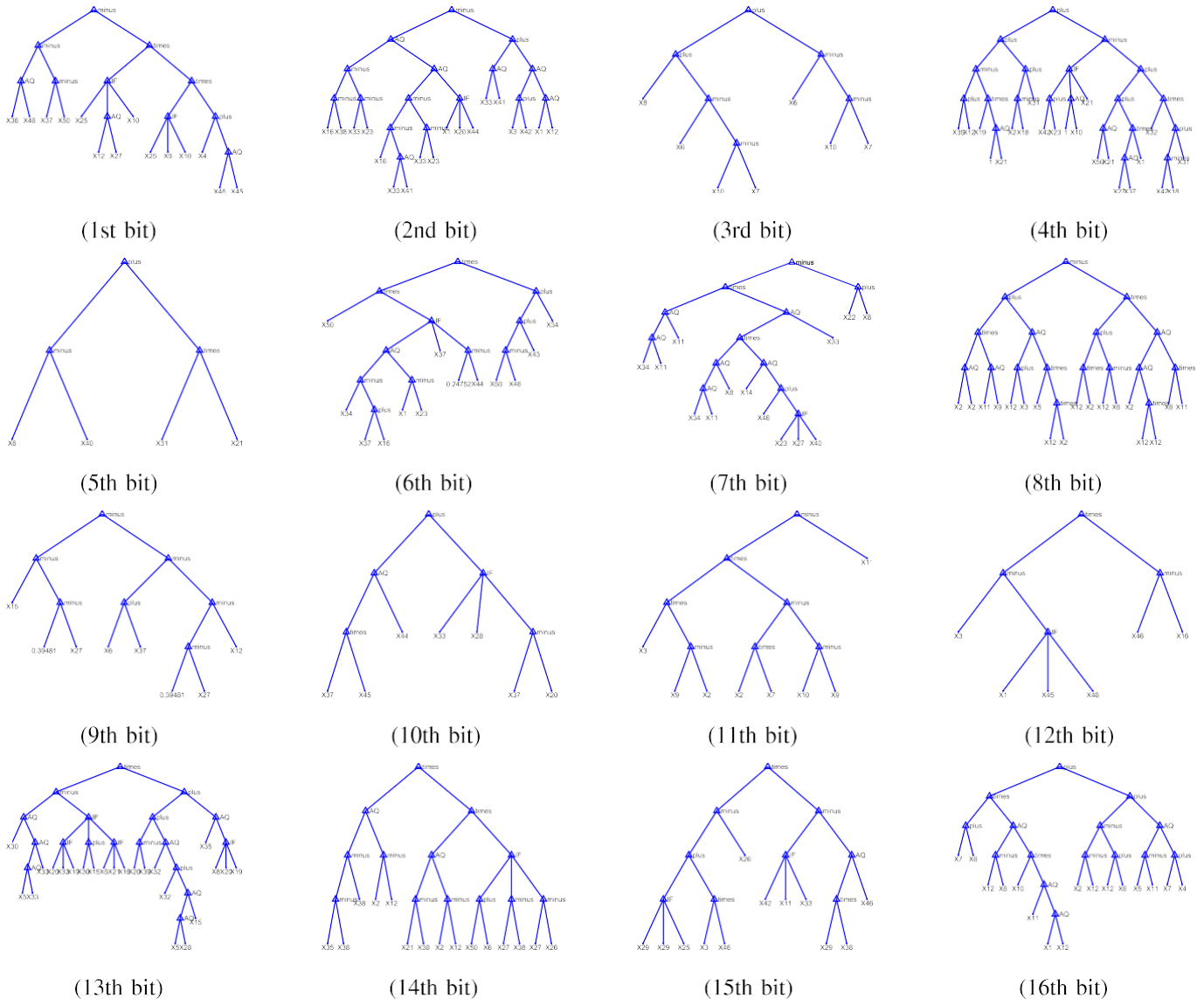


Fig. 7. GP-evolved tree hash functions for embedding 8-b binary codes on the SIFT 1M data set. From top-left to bottom-right, each tree illustrates a hashing function for a bit. The nodes "plus," "minus," "times," and "AQ" in the tree correspond with "+," "−," "×," and "÷," respectively.

function on a 128-D SIFT feature. We first assume that the 6th value (X6) in the 128-SIFT feature is 0.231, the 40th value (X40) is 0.642, the 31st value (X31) is 0.973, and the 21st value (X21) is 0.156. As shown in Fig. 7, we compute the output value of the fifth bit hash function as $(X6 - X40) + (X31 \times X21) = (0.231 - 0.642) + (0.937 \times 0.156) = -0.265$. Thus, the fifth bit is binary $(-0.265) = 0$. Without loss of generality, the remaining bits generated by the GP-tree-based hash functions can also be computed similar to the above procedure.

We also list the training time and the test time for different algorithms on two data sets in Table III. Considering the training time, the random projection-based algorithms are relatively more efficient, especially the LSH. Our basic ECE is time-consuming but gives significantly better accuracies. In this paper, our target is to obtain better results by slightly sacrificing the efficiency. Thus, our proposed basic ECE focuses more on the improvement of accuracies rather than time complexity. To make our method more scalable for large-scale tasks, we have also upgraded our ECE into a more efficient version named RBPL-ECE, which can significantly improve the efficiency. From Table III, our RBPL-ECE also produces better results than all other compared methods in terms of the retrieval accuracies. In particular, RBPL-ECE can perform more efficiently in the training phase but still yields better results than popular hashing methods such as STH, KSH, LLH, SpherH, BSSC, and RBM. It is noteworthy that once the optimal hashing functions of our method are obtained from the training phase, the optimized hashing functions will be fixed and directly used for new data. In addition, with the rapid development of silicon technologies, future computers will be much faster and even the training will become less a problem. In terms of the test phase, LSH and PCAH are the most efficient methods. Both of them simply need a matrix multiplication and a thresholding to obtain the binary codes. Due to the extra concatenation step in our parallel coding,

TABLE IV

COMPARISON OF PERFORMANCE AND TIME COMPLEXITY RATIO ON SIFT 1M WITH 48 b

Compared methods	Precision increment ratio	Train time decrement ratio	Test time decrement ratio
CH Vs. RBPL-ECE	7.502%	-902.3%	38.41%
ITQ Vs. RBPL-ECE	21.31%	-9.82%	6.16%
SpherH Vs. RBPL-ECE	17.27%	57.21%	12.07%
KSH Vs. RBPL-ECE	14.21%	12.14%	10.42%
STH Vs. RBPL-ECE	21.69%	45.33%	-69.19%

The values colored by red indicate our method, i.e., RBPL-ECE, is better than compared methods and the green values indicate worse performance than compared methods. The ratio is calculated as the following example: if the precision of CH is 0.175 and the precision of RBPL-ECE is 0.204, the precision increment ratio (CH Vs. RBPL-ECE) is equal to $\frac{0.204-0.175}{0.175} \times 100\% = 16.57\%$.

TABLE V

COMPARISON OF PERFORMANCE AND TIME COMPLEXITY RATIO ON GIST 1M WITH 32 b

Compared methods	Precision increment ratio	Train time decrement ratio	Test time decrement ratio
CH Vs. RBPL-ECE	16.57%	-776.2%	48.98%
ITQ Vs. RBPL-ECE	41.67%	-26.86%	3.25%
SpherH Vs. RBPL-ECE	29.11%	11.46%	9.67%
KSH Vs. RBPL-ECE	50.00%	34.61%	4.39%
STH Vs. RBPL-ECE	65.85%	10.53%	-53.52%

The values colored by red indicate our method, i.e., RBPL-ECE, is better than compared methods and the green values indicate worse performance than compared methods. The ratio is calculated as the following example: if the precision of CH is 0.175 and the precision of RBPL-ECE is 0.204, the precision increment ratio (CH Vs. RBPL-ECE) is equal to $\frac{0.204-0.175}{0.175} \times 100\% = 16.57\%$.

RBPL-ECE is slightly slower than the (basic) ECE for testing. AGH and SpH are the most expensive methods for testing, due to the relatively high cost when calculating the sparse representation and computing the analytical eigenfunctions, respectively. To further illustrate the effectiveness of our methods, we also illustrate the performance and time complexity ratio on both the data sets in Tables IV and V compared with the five best performed hashing techniques. The results indicate that RBPL-ECE can lead to better performance compared with SpherH and KSH in terms of precision, training time, and test time. Compared with CH and ITQ, RBPL-ECE still achieves superior performance and test time, but more time is needed in the training phase. STH is the most efficient one in the querying phase among all the six methods. From the overall tendency in Tables IV and V, our RBPL-ECE produces a better tradeoff between precision and time complexity.

VI. CONCLUSION

In this paper, we have presented a novel unsupervised hashing framework, ECE, to learn highly discriminative binary codes for large-scale similarity search. It is addressed as an optimization problem that combines GP with the boosting-based weight updating trick. For each bit of ECE, the proposed learning scheme evolves a weak binary classification function through GP and reweights the training samples for the next bit to jointly minimize its empirical risk with the AdaBoost strategy. To further reduce the computational complexity, we improved the basic ECE using the RBPL technique. It is demonstrated that RBPL is more efficient for large-scale training but can still achieve competitive results. Two standard data sets SIFT 1M and GIST 1M have been systematically evaluated, and show promising results compared with the state-of-the-art hashing methods. In the future work, we will focus on optimizing our ECE algorithm to make it more compact and discovering more efficient seeding solutions to initialize GP instead of using the random scheme.

REFERENCES

[1] A. Gionis *et al.*, "Similarity search in high dimensions via hashing," in *Proc. Int. Conf. Very Large Data Bases*, Sep. 1999, pp. 518–529.

[2] B. Kulis and K. Grauman, "Kernelized locality-sensitive hashing," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 6, pp. 1092–1104, Jun. 2012.

[3] L. Liu, M. Yu, and L. Shao, "Projection bank: From high-dimensional data to medium-length binary codes," in *Proc. Int. Conf. Comput. Vis.*, Santiago, Chile, Dec. 2015, pp. 1–8.

[4] Y. Gong and S. Lazebnik, "Iterative quantization: A procrustean approach to learning binary codes," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Colorado Springs, CO, USA, Jun. 2011, pp. 817–824.

[5] J.-P. Heo, Y. Lee, J. He, S.-F. Chang, and S.-E. Yoon, "Spherical hashing," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Providence, RI, USA, Jun. 2012, pp. 2957–2964.

[6] W. Liu, J. Wang, S. Kumar, and S.-F. Chang, "Hashing with graphs," in *Proc. Int. Conf. Mach. Learn.*, Washington, DC, USA, Jun. 2011, pp. 1–8.

[7] R. Salakhutdinov and G. Hinton, "Semantic hashing," *Int. J. Approx. Reasoning*, vol. 50, no. 7, pp. 969–978, Jul. 2009.

[8] L. Liu, M. Yu, and L. Shao, "Multiview alignment hashing for efficient image search," *IEEE Trans. Image Process.*, vol. 24, no. 3, pp. 956–966, Mar. 2015.

[9] F. Zhu and L. Shao, "Weakly-supervised cross-domain dictionary learning for visual recognition," *Int. J. Comput. Vis.*, vol. 109, no. 1, pp. 42–59, Aug. 2014.

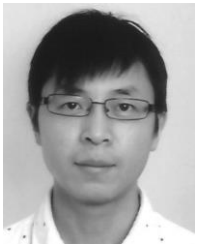
[10] L. Liu, M. Yu, and L. Shao, "Unsupervised local feature hashing for image similarity search," *IEEE Trans. Cybern.*, doi: 10.1109/TCYB.2015.2480966.

[11] V. Gaede and O. Günther, "Multidimensional access methods," *ACM Comput. Surv.*, vol. 30, no. 2, pp. 170–231, Jun. 1998.

[12] G. Shakhnarovich, "Learning task-specific similarity," Ph.D. dissertation, Dept. Elect. Eng. Comput. Sci., Massachusetts Inst. Technol., Cambridge, MA, USA, 2005.
[13] J. Wang, S. Kumar, and S.-F. Chang, "Semi-supervised hashing for large-scale search," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 12, pp. 2393–2406, Dec. 2012.
[14] Y. Weiss, A. Torralba, and R. Fergus, "Spectral hashing," in *Proc. Neural Inf. Process. Syst.*, Vancouver, BC, Canada, Dec. 2008, pp. 1753–1760.
[15] D. Zhang, J. Wang, D. Cai, and J. Lu, "Self-taught hashing for fast similarity search," in *Proc. SIGIR*, Geneva, Switzerland, Jul. 2010, pp. 18–25.
[16] Z. Cai, L. Liu, M. Yu, and L. Shao, "Latent structure preserving hashing," in *Proc. Brit. Mach. Vis. Conf.*, Swansea, U.K., Sep. 2015, pp. 546–556.
[17] Y. Lin, R. Jin, D. Cai, S. Yan, and X. Li, "Compressed hashing," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Portland, OR, USA, Jun. 2013, pp. 446–451.
[18] R. Poli, W. W. B. Langdon, N. F. McPhee, and J. R. Koza, *A Field Guide to Genetic Programming*. Raleigh, NC, USA: Lulu Publishing Company, 2008.
[19] L. Liu, L. Shao, and P. Rockett, "Genetic programming-evolved spatio-temporal descriptor for human action recognition," in *Proc. Brit. Mach. Vis. Conf.*, Surrey, U.K., Sep. 2012, pp. 1–12.
[20] L. Shao, L. Liu, and X. Li, "Feature learning for image classification via multiobjective genetic programming," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 7, pp. 1359–1371, Jul. 2014.
[21] L. Liu and L. Shao, "Learning discriminative representations from RGB-D video data," in *Proc. 23rd Int. Joint Conf. Artif. Intell.*, Beijing, China, Aug. 2013, pp. 1493–1500.
[22] J. Friedman, T. Hastie, and R. Tibshirani, "Additive logistic regression: A statistical view of boosting," *Ann. Statist.*, vol. 28, no. 2, pp. 337–407, 2000.
[23] G. Shakhnarovich, P. Viola, and T. Darrell, "Fast pose estimation with parameter-sensitive hashing," in *Proc. Int. Conf. Comput. Vis.*, Nice, France, Dec. 2003, pp. 750–757.
[24] A. Torralba, R. Fergus, and Y. Weiss, "Small codes and large image databases for recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Anchorage, AK, USA, Jun. 2008, pp. 1–8.
[25] T. Trzcinski, M. Christoudias, V. Lepetit, and P. Fua, "Learning image descriptors with the boosting-trick," in *Proc. Neural Inf. Process. Syst.*, Harrahs and Harveys, NV, USA, Dec. 2012, pp. 269–277.
[26] T. Trzcinski, M. Christoudias, P. Fua, and V. Lepetit, "Boosting binary keypoint descriptors," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Portland, OH, USA, Jun. 2013, pp. 2874–2881.
[27] L. Liu, L. Shao, and X. Li, "Evolutionary compact embedding for large-scale image classification," *Inf. Sci.*, vol. 316, pp. 567–581, Sep. 2015.
[28] C. Leng, J. Cheng, J. Wu, X. Zhang, and H. Lu, "Supervised hashing with soft constraints," in *Proc. ACM Int. Conf. Inf. Knowl. Manage.*, Shanghai, China, Nov. 2014, pp. 1851–1854.
[29] L. Liu, M. Yu, and L. Shao, "Local feature binary coding for approximate nearest neighbor search," in *Proc. Brit. Mach. Vis. Conf.*, Swansea, U.K., Sep. 2015, pp. 132–143.
[30] J. Qin, L. Liu, M. Yu, Y. Wang, and L. Shao, "Fast action retrieval from videos via feature disaggregation," in *Proc. Brit. Mach. Vis. Conf.*, Swansea, U.K., Sep. 2015, pp. 341–351.
[31] W. Liu, C. Mu, S. Kumar, and S.-F. Chang, "Discrete graph hashing," in *Proc. Neural Inf. Process. Syst.*, Montreal, QC, Canada, Dec. 2014, pp. 3419–3427.
[32] J. Ni, R. H. Drieberg, and P. I. Rockett, "The use of an analytic quotient operator in genetic programming," *IEEE Trans. Evol. Comput.*, vol. 17, no. 1, pp. 146–152, Feb. 2013.
[33] U. Bhowan, M. Zhang, and M. Johnston, "Genetic programming for image classification with unbalanced data," in *Proc. Int. Conf. Image Vis. Comput. New Zealand*, Queenstown, New Zealand, Nov. 2009, pp. 316–321.
[34] D. P. Muni, N. R. Pal, and J. Das, "A novel approach to design classifiers using genetic programming," *IEEE Trans. Evol. Comput.*, vol. 8, no. 2, pp. 183–196, Apr. 2004.
[35] J. Eggermont, J. N. Kok, and W. A. Kosters, "Genetic programming for data classification: Partitioning the search space," in *Proc. ACM Symp. Appl. Comput.*, Nicosia, Cyprus, Mar. 2004, pp. 1001–1005.
[36] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, vol. 1. Cambridge, MA, USA: MIT Press, 1992.
[37] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Kauai, HI, USA, Jun. 2001, pp. I-511–I-518.
[38] J. Wang, S. Kumar, and S.-F. Chang, "Sequential projection learning for hashing with compact codes," in *Proc. Int. Conf. Mach. Learn.*, Haifa, Israel, Jun. 2010, pp. 1127–1134.
[39] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.
[40] W. Liu, J. Wang, R. Ji, Y.-G. Jiang, and S.-F. Chang, "Supervised hashing with kernels," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Providence, RI, USA, Jun. 2012, pp. 2074–2081.
[41] G. Irie, Z. Li, X.-M. Wu, and S.-F. Chang, "Locally linear hashing for extracting non-linear manifolds," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Columbus, OH, USA, Jun. 2014, pp. 2123–2130.

Li Liu received the B.Eng. degree in electronic information engineering from Xi'an Jiaotong University, Xi'an, China, in 2011, and the Ph.D. degree from the Department of Electronic and Electrical Engineering, University of Sheffield, Sheffield, U.K., in 2014.

He is currently a Research Fellow with the Department of Computer Science and Digital Technologies, Northumbria University, Newcastle upon Tyne, U.K. His current research interests include computer vision, machine learning, and data mining.



Ling Shao (M'09–SM'10) is a Professor with the Department of Computer Science and Digital Technologies at Northumbria University, Newcastle upon Tyne, U.K. Previously, he was a Senior Lecturer (2009-2014) with the Department of Electronic and Electrical Engineering at the University of Sheffield and a Senior Scientist (2005-2009) with Philips Research, Eindhoven, The Netherlands. His research interests include Computer Vision, Image/Video Processing and Machine Learning. He is an associate editor of IEEE TRANSACTIONS ON IMAGE PROCESSING, IEEE TRANSACTIONS ON CYBERNETICS, and several other journals. He is a Fellow of the British Computer Society and the Institution of Engineering and Technology.