Network Flow Formulations for Learning Binary Hashing

Lopamudra Mukherjee[1(✉)], Jiming Peng[2], Trevor Sigmund[1], and Vikas Singh[3]

[1] University of Wisconsin–Whitewater, Whitewater, USA
{mukherjl,sigmundTJ27}@uww.edu
[2] University of Houston, Houston, USA
jopeng@central.uh.edu
[3] University of Wisconsin–Madison, Madison, USA
vsingh@biostat.wisc.edu

Abstract. The problem of *learning binary hashing* seeks the identification of a binary mapping for a set of n examples such that the corresponding Hamming distances preserve high fidelity with a *given* $n \times n$ matrix of distances (or affinities). This formulation has numerous applications in efficient search and retrieval of images (and other high dimensional data) on devices with storage/processing constraints. As a result, the problem has received much attention recently in vision and machine learning and a number of interesting solutions have been proposed. A common feature of most existing solutions is that they adopt continuous iterative optimization schemes which is then followed by a post-hoc rounding process to recover a feasible discrete solution. In this paper, we present a fully combinatorial network-flow based formulation for a relaxed version of this problem. The main maximum flow/minimum cut modules which drive our algorithm can be solved efficiently and can directly learn the binary codes. Despite its simplicity, we show that on most widely used benchmarks, our proposal yields competitive performance relative to a suite of *nine* different state of the art algorithms.

1 Introduction

The need to perform quick indexing and retrieval on large collections of images and videos, both in our personal collections and on the web, has led to a resurgence of interest in efficient methods for hashing.

While the literature on this topic is quite mature, the need to accomplish retrieval tasks on novel mobile architectures has led to numerous interesting variations of the problem. For example, small form factor devices with limited storage capacity as well as the associated power consumption constraints typically involve unique economy/accuracy trade-offs for the algorithm. Separately, when the data correspond to images, notice that it may not be sensible to hash the native image representation expressed as a $\mathbb{R}^D$ vector. Indeed, much of the image may include background clutter as well as other content not directly pertinent to the semantic information of the image. This leads to interesting variations of hashing

B. Leibe et al. (Eds.): ECCV 2016, Part V, LNCS 9909, pp. 366–381, 2016.
DOI: 10.1007/978-3-319-46454-1_23

where each to-be-hashed example may actually correspond to multiple patches (or salient objects) contained within an image.

In vision applications of hashing, it is often the case that rather than represent the image in terms of its components as described above, it is more convenient to compute similarities between pairs of images in the dataset [1,2]. In practice, this is accomplished by running an object detector on each image and assessing whether each pair of images contain similar content. For instance, we may assume that images where detectors find similar objects are likely to be similar. So, for a given corpus of n images, we obtain a $n \times n$ similarity matrix via standard pre-processing. The hashing problem here requires finding a lower dimensional (say, d) mapping for each example in the corpus such that distances (or similarities) in the embedded space are low-distortion, relative to the original similarities. The "binary" version of this problem instead seeks an embedding to a d-dimensional $\{0, 1\}$ space — this formulation is called Binary hashing. There is much recent interest in this problem due to two key reasons. First, binary hashing makes the downstream retrieval task extremely easy. For instance, the Hamming distance between a query and an item in the database can now be computed simply via logical operations (e.g., XOR) which leads to efficiency advantages. Second, the storage requirements are modest: even large image datasets such as *ImageNet* can easily be stored on commodity handheld devices without compression.

Deriving a binary embedding which maintains good fidelity with a given similarity matrix is commonly known in the literature as *learning* binary hashing. There is a growing body of work in computer vision and machine learning on various nice approaches to the problem. Note that the core problem is NP-hard (both non-convex and discrete), so most proposed methods involve iterative *continous* optimization schemes or make use of various other spectral relaxations that need solutions to large eigen-value problems [3], coordinate descent [4], Procrustean quantization [5] and concave-convex optimization [6]. These techniques often produce a relaxed solution, which is then 'rounded' based on its sign. These are effective approaches and work well in many cases though approximation ratios may be difficult to obtain. Our goal is to investigate, via an interesting reformulation, the feasibility of (arguably) simpler *discrete* energy minimization algorithms for a relaxed version of this problem and to assess if it can still offer competitive performance. In particular, we seek to derive fully combinatorial algorithms for learning binary hashing. Recall that while combinatorial methods have various benefits and dominate the landscape of image segmentation in vision, one attractive feature is that the primitive operations required (e.g., addition, counting) are simple. This yields substantial efficiency benefits for our main training module and is appropriate in situations where support for more intensive low-level operations is unavailable or otherwise undesirable. We present here a network-flow based formulation that is easy to implement and yields a *provably* partially-optimal solution for each bit of the hash code. To our knowledge, no combinatorial approaches currently exist for learning binary hashing in an unsupervised setup using network flow. Further, our approach extends to any arbitrary $\|.\|_p^p$ norm, though in this paper, for presentation purposes, we restrict our treatment to ℓ_1-norms and sum of squares distances.

Related Work. A well known algorithm for hashing is Locality-Sensitive Hashing (LSH) [7]. LSH creates a hashing scheme where similar items fall in the same bucket with high probability. Such embeddings are obtained via projections on randomly generated hyperplanes, where the query time for a $(1+\epsilon)$ approximate nearest neighbor can be bounded by $O(n^{\frac{1}{1+\epsilon}})$. An advantage of LSH is that the random projections provably preserve distances in the limit as the number of hash bits increases. But it has been observed that the number of hash bits required may need to be large to faithfully maintain distances. More recent work has focused on better exploiting the structure of the data in specific domains by generating more effective/tailored hash functions. To this [8] end, within vision and machine learning, a number of methods [6] have been proposed including: Semantic Hashing [9,10], Spectral Hashing (SH) [3], Kernelized Spectral Hashing (KLSH) [11], Multi-dimension Spectral Hashing (MDSH) [12], Iterative Quantization (ITQ) [5], BRE [4], Anchor Graph Hashing (AGH) [8,13], NMF based Hashing [14], Self-taught Hashing (STH) [15], and Fast Hash [16]. Other proposals include parameter sensitive hashing [17] and asymmetric hash functions [18]. We briefly review some existing discrete formulations of the hashing problem. Most of these methods focus on the supervised version of the problem. A graph partitioning approach has been used in the supervised case by [19]. For the same problem, [20] employs an approach that solves the NP-hard hashing problem as regularized sub-problems that admits an analytical solution via cyclic coordinate descent. In terms of the unsupervised problem, to the best of our knowledge, [8] is the only approach that uses a discrete optimization based method for graph hashing using the anchor graph method. It proposes a bilevel maximization, one of which can be solved in closed form. It should be noted that the objective and subsequent optimization techniques in [8] are not similar to the network-flow approach proposed in this paper. As we will describe shortly, our proposal is also among a few hashing based approaches such as [21], that uses a simple two stage process for evaluating unseen data: first, we generate classifiers using the training code as labels and later these classifiers are used to predict the hashing code for test examples. Our experiments show that this scheme, though not widely utilized currently, has merit in vision applications.

2 Setting Up the Optimization Model

In this section, we will setup a simple optimization model that expresses the key properties of a desired solution to the learning binary hashing problem. Then, we will describe the details of our proposed algorithm and its analysis.

Notations. We use upper case letters such as Q to denote matrices and bold lower case characters such as $\mathbf{q}$ for vectors. When referring to a set of vectors, the ith vector will be $\mathbf{q}_i$; the jth entry of a vector $\mathbf{q}$ will be denoted as q_i.

Let $Y \in \mathbb{R}^{n \times D}$ be a matrix comprised of n examples in D dimensions as its rows. Let $A \in \mathbb{R}^{n \times n}$ be the matrix of pairwise similarities of examples in Y. Our goal is to find an embedding of Y into a lower dimensional binary space $X \in \{0,1\}^{n \times d}$, where $D \gg d$. Clearly, the above embedding should preserve

the relative distances between the n examples. Since X is binary, we use the Hamming distance between the examples in X. Let $H \in \mathbb{Z}^{n \times n}$ be the matrix of the corresponding pairwise Hamming distances, i.e., each entry H_{ij} computes the Hamming distance between examples i and j in X. The objective is given as

$$\arg\min_{X} d_f(A - H) \quad \text{s.t.} \quad H = \Gamma(X), \tag{1}$$

where $d_f(\cdot)$ is an appropriate loss function, e.g., capturing the distance between matrices, and the constraint involving $\Gamma(\cdot)$ asks that H must capture the Hamming distances between the examples in the (embedded) binary space.

Let us now define Γ by writing H as a function of X. This can be expressed as $H_{ij} = \sum(\mathbf{x}_{i-} \oplus \mathbf{x}_{j-})$ where $\mathbf{x}_{i-}$ and $\mathbf{x}_{j-}$ are rows of X (i.e., the binary embedding of examples i and j) and $\oplus$ denotes the bitwise XOR operation of two binary vectors. But substituting this format directly into (1) is not very useful. Instead, we can observe that the corresponding XOR operation can be written in quadratic form as

$$H = X(\mathbf{1} - X)^T + (\mathbf{1} - X)X^T, \tag{2}$$

where $\mathbf{1}$ is a matrix of 1s of appropriate size. This formulation, when substituted in (1), unfortunately, does not lead to tractable forms for most choices of $d_f(.)$. But interestingly, as we will see shortly, if the loss function is the *Sum of Square Distances (SSD)* and/or the ℓ_1 distance, then we obtain models with distinct advantages. We focus on these two examples for the loss since they are widely used but our solution extends to arbitrary $\|.\|_p^p$ distances.

3 Minimizing the Sum of Squares Distances

In this section, we first describe how problem (1) can be reformulated as a network flow when the loss function is a sum of squares distance (SSD). After presenting the model, we will give an efficient sequential update strategy.

Let $1_{n \times d}$ denote a matrix of 1s. Let us rewrite (1) to derive an identity for H,

$$\begin{aligned} H =& X(1_{n\times d} - X)^T + (1_{n\times d} - X)X^T \\ =& -(1_{n\times d} - X)(1_{n\times d} - X)^T - XX^T + d1_{n\times n} = \quad -\bar{X}\bar{X}^T - XX^T + d1_{n\times n} \end{aligned} \tag{3}$$

Let $\bar{X}$ denote the term $1_{n\times d} - X$. Immediately, we have that

$$A - H = \bar{X}\bar{X}^T + XX^T - (d1_{n\times n} - A) = \bar{X}\bar{X}^T + XX^T - \bar{A}, \tag{4}$$

where $\bar{A} = d1_{n\times n} - A$. Now, we rewrite problem (1) using the SSD measure,

$$\min_X \|\bar{A} - \bar{X}\bar{X}^T - XX^T\|_2^2 = \min_X \|\bar{A} - \sum_{i=1}^{d}(\mathbf{x}_i\mathbf{x}_i^T + \bar{\mathbf{x}}_i\bar{\mathbf{x}}_i^T)\|_2^2, \tag{5}$$

where $\mathbf{x}_i, \bar{\mathbf{x}}_i$ are the i-th columns of the binary matrices X and $\bar{X}$ respectively. Observe that by definition, we have

$$\mathbf{x}_i + \bar{\mathbf{x}}_i = 1_{n\times 1}, \qquad \mathbf{x}_i^T\bar{\mathbf{x}}_i = 0, \qquad \forall i = 1, \cdots, d. \tag{6}$$

3.1 Bit-by-Bit Network Flow

Solving Problem (5) for all d columns of X concurrently turns out to be difficult. So, we adopt a sequential strategy here — we temporarily fix $d-1$ columns of the matrix X and update a single column $\mathbf{x}_{\text{iter}}$ at a time. To reduce notational clutter, instead of $\mathbf{x}_{\text{iter}}$, we will drop the subscript and $\mathbf{x}$ will denote the column currently being updated. All other columns will be referred to as $\mathbf{x}_{i|i\neq\text{iter}}$.

To optimize the entries relevant to a single bit iter, we start with (5) and take out terms for all 'frozen' (for this update) entries $i \neq$ iter.

$$\min_{X} || \underbrace{\bar{A} - \sum_{i=1|i\neq\text{iter}}^{d} (\mathbf{x}_i\mathbf{x}_i^T + \bar{\mathbf{x}}_i\bar{\mathbf{x}}_i^T)}_{G} - (\hat{\mathbf{x}}\hat{\mathbf{x}}^T + \hat{\bar{\mathbf{x}}}\hat{\bar{\mathbf{x}}}^T)||_2^2.$$

Letting $G = \bar{A} - \sum_{i=1,i\neq\text{iter}}^{d}(\hat{\mathbf{x}}_i\hat{\mathbf{x}}_i^T + \hat{\bar{\mathbf{x}}}_i\hat{\bar{\mathbf{x}}}_i^T)$ is convenient because we can now consider G to be the weight matrix associated with a network (or an undirected graph) of n nodes, i.e., $N = \{v_1, \cdots, v_n\}$ where every entry G_{ij} denotes the weight of the edge from v_i to v_j. Here, we also have that the to-be-updated $\mathbf{x} \in \{0,1\}^n$ and $\bar{\mathbf{x}} = 1_{n\times 1} - \mathbf{x}$. With these definitions, we can formally write down the following subproblem which needs to be solved in any update step,

$$\min \|G - (\mathbf{x}\mathbf{x}^T + \bar{\mathbf{x}}\bar{\mathbf{x}}^T)\|_2^2, \tag{7}$$

A Potential Solution via Minimum Cuts. For given binary $\mathbf{x}$ and $\bar{\mathbf{x}}$, let us define two subsets of nodes as follows.

$$N_1 = \{v_i : x_i = 1\}, \quad \bar{N}_1 = N - N_1 = \{v_i : \bar{x}_i = 1\}.$$

Denote the entry-wise square of the matrix, G as G_{ij}^2. We call $\widehat{G}$ the difference matrix whose elements are defined as $\widehat{G}_{ij} = G_{ij}^2 - (G - 1_{n\times n})_{ij}^2$. By inspection,

$$\begin{aligned} \mathbf{G}(\mathbf{x}) &= \|G - (\mathbf{x}\mathbf{x}^T + \bar{\mathbf{x}}\bar{\mathbf{x}}^T)\|_2^2 &(8)\\ &= \sum_{i,j}(G - 1_{n\times n})_{ij}^2 + 2\sum_{i\in N_1, j\in\bar{N}_1}\widehat{G}_{ij}. &(9)\end{aligned}$$

Properties of (9). The identity in (9) has a particularly interesting form. First, note that the first term is constant. Now, if we think of a graph whose edge weights are given by the matrix $\widehat{G}$, then by inspection, we see that the second term precisely minimizes the cut edges of that graph – the resultant partitions induced by the cut are N_1 and $\bar{N}_1$. This means that our objective has reduced to the classic formulation of network flow on graphs. Therefore, solving problem (7) is *equivalent* to finding the minimum cut in a network with a weight matrix $\widehat{G}$ (see footnote). Since the minimum cut problem can be solved effectively in polynomial time [22], we immediately have the following result.

Theorem 1. *Problem* (7) *is poly-time solvable if edge weights are non-negative*[1].

4 Formulating the ℓ_1 Distance Setting

In this section, we will derive an analogous solution scheme when the loss is ℓ_1 distance, natural in various applications. Most steps here will be identical to the SSD setup. However, we will need some additional constraints to derive meaningful solutions for the motivating application, which will require special treatment (to be optimizable combinatorially). Using ℓ_1 loss, analogous to (5),

$$\min_X \|\bar{A} - \bar{X}\bar{X}^T - XX^T\|_1 = \qquad \min_X \|\bar{A} - \sum_{i=1}^{d}(\mathbf{x}_i\mathbf{x}_i^T + \bar{\mathbf{x}}_i\bar{\mathbf{x}}_i^T)\|_1, \tag{10}$$

where the constraints in (6) hold, as in the SSD case. As before, to solve problem (10) sequentially, we fix $d-1$ columns of the matrix X and update a single column $\mathbf{x}$ to reduce the objective. Analogous to (7) earlier, this leads to,

$$\min_X \|G - (\mathbf{x}\mathbf{x}^T + \bar{\mathbf{x}}\bar{\mathbf{x}}^T)\|_1, \tag{11}$$

where G is again the weight matrix associated with a network (or an undirected graph) of n nodes ($N = \{v_1, \cdots, v_n\}$), and every entry G_{ij} denotes the weight of the edge from v_i to v_j. Further, $\mathbf{x} \in \{0,1\}^n, \bar{\mathbf{x}} = 1_{n\times 1} - \mathbf{x}$. We next briefly discuss a reformulation as a network flow.

For binary $\mathbf{x}$ and $\bar{\mathbf{x}}$, we define two subsets of nodes as

$$N_1 = \{v_i : x_i = 1\}, \quad \overline{N}_1 = N - N_1 = \{v_i : \bar{x}_i = 1\}.$$

Define $\widetilde{G} = |G| - |G - 1_{n\times n}|$ where $|M|$ denotes the entry-wise absolute value of the matrix. We can easily see that

$$\mathbf{G}(\mathbf{x}) = \|G - (\mathbf{x}\mathbf{x}^T + \bar{\mathbf{x}}\bar{\mathbf{x}}^T)\|_1 \quad = \sum_{i,j} |G - 1_{n\times n}|_{ij} + 2 \sum_{i\in N_1, j\in \overline{N}_1} \widetilde{G}_{ij}. \tag{12}$$

Properties of (12). The properties are analogous to those of (9). Therefore, solving (11) is equivalent to finding the network flow with a weight matrix $\widetilde{G}$. All advantages (and limitations) of the SSD setting extend to the ℓ_1 case as well.

5 Constraints for Balanced Partitions

A network flow formulation for the above model has significant computational benefits. Unfortunately, a disadvantage of the formulation is that it may lead to

[1] This requires $G \geq \frac{1}{2}$ entry- wise which can be ensured by a scaling of the input matrix A. However, this scaling is not utilized, since we eventually derive an extended model that generalizes better and needs a QPBO-type solver due to the issues highlighted later in Sect. 5. That formulation is not a min-cut and can handle negative weights.

disproportionate cuts – that is, one partition is small (with only a few nodes), whereas the other partition is large. When the model returns such solutions (for one or more bits), expectedly, it will adversely affect the quality of the resultant hashing. This issue is not specific to our proposal; in minimum cuts based graph partitioning schemes, this behavior has been reported by [23] and others: the so-called "shrinking" bias, i.e., favoring cutting small sets of isolated nodes in the graph. On the other hand, as observed in [3] and other works, a desired property of hashing functions is that ideally each bit has a 50 % chance of being one or zero. This means that, in expectation, our model should return almost size balanced partitions. In other words, we need to regularize our formulation to prefer cuts that are almost size balanced. This is known as the Balanced Cut problem [24]. When formulated via hard constraints, the goal is to ensure that the size of each partition $\geq \beta n$, where $\beta \leq .5$ is a constant and n is a number of nodes in the graph. This problem is NP-hard. While various relaxations exist [25], many are expensive to optimize and difficult to incorporate within our network flow formulation. Our formulation does not have a specific need for a hard size constraint; instead, we ask that the disbalance in the two partition sizes must be penalized as a term in the objective. This leads to a Quadratic Pseudoboolean function (QPB), which can still be solved using network flow.

5.1 Size Regularized Cuts

Let $\mathbf{x} \in \{0,1\}^n$ be the solution from a bit-level mincut problem. Since there are only two partitions, i.e., $x_i = 0$ or $x_i = 1$, size balance requires that the values of $\sum_{i=1}^{n} x_i$ and $\sum_{i=1}^{n}(1 - x_i)$ be close. When normalized, this implies

$$\frac{1}{n}\sum_{i=1}^{n} x_i \approx \frac{1}{n}\sum_{i=1}^{n}(1 - x_i) \approx \frac{1}{2}. \tag{13}$$

Let $p = \sum_{i=1}^{n} x_i$ and $q = \sum_{i=1}^{n}(1 - x_i)$ where we have $p+q = n$, i.e., each term p (and q) counts the number of nodes in each partition. We can verify that $pq \leq \frac{n^2}{4}$ and this inequality becomes tight only if $p = q$. Therefore, an optimal balance is achieved when pq is maximized or when the value of $p^2 + q^2$ is minimized. This observation yields a reformulated balance regularization term. We see

$$p^2 = \mathbf{x}\mathbf{1}_{n\times n}\mathbf{x}^T, \quad q^2 = (1 - \mathbf{x})\mathbf{1}_{n\times n}(1 - \mathbf{x})^T. \tag{14}$$

So, the size balance regularization can be achieved via minimizing,

$$\mathbf{R}(\mathbf{x}) = \mathbf{x}\mathbf{1}_{n\times n}\mathbf{x}^T + (1 - \mathbf{x})\mathbf{1}_{n\times n}(1 - \mathbf{x})^T$$

Alternatively, via rearranging the above formulation, we can also minimize,

$$\begin{aligned}\mathbf{R}(\mathbf{x}) &= \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n} x_i x_j + \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}(1 - x_i)(1 - x_j) \\ &= \frac{2}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n} x_i x_j - \frac{2}{n}\sum_{i=1}^{n} x_i + \text{const}\end{aligned} \tag{15}$$

5.2 Reparameterization and Graph Construction

Putting together the above expression with the main network flow model, we will solve following model for each bit,

$$\min_{\mathbf{x}} E(\mathbf{x}) = \mathbf{G}(\mathbf{x}) + \alpha \mathbf{R}(\mathbf{x}) \tag{16}$$

where α is the user defined influence of the regularizer.

Properties of (16). The model in (16) is the so-called Quadratic Pseudoboolean form [26–28], which consists of quadratic and linear terms. Functions of this form can be solved as a network flow model on a specially constructed graph. We will shortly discuss the specifics of graph construction for this task. We will represent each variable x_i as a pair of literals, y_i and $\bar{y}_i$ where $\bar{y}_i$ will represent $1 - x_i$. This pair of literals will correspond to a pair of nodes in a to-be-constructed graph $\mathcal{G}$. Edges will be added to $\mathcal{G}$ based on various terms in the corresponding QPB, to be discussed shortly. The partition computed on $\mathcal{G}$ will determine the assignments of variables x_i to 1 (or 0). Depending on how the nodes in $\mathcal{G}$ for a pair of literals y_i and $\bar{y}_i$ are partitioned, we will either get "persistent" integral 0/1 solutions for x_i (provably consistent with the optimal) or have $\frac{1}{2}$ (half integral) values assigned to x_i which will need additional rounding to obtain a $\{0, 1\}$ solution. This is the reason QPB solution are considered *partially optimal* solutions — i.e., entries in the solution which are integral are known to be optimal.

Reparameterization. Before describing the construction of $\mathcal{G}$, we will reparameterize the coefficients in our objective as a vector $\mathbf{\Phi}$. More specifically, we will rewrite the energy by collecting the unary and pairwise costs in (16) as the coefficients of the linear and quadratic variables, denoted by Φ_i and Φ_{ij} respectively. The unary cost is set as $\Phi_i = -\frac{2\alpha}{n}$. Other unary functions can also be included here for stronger guidance. Each pairwise cost can be attributed to one of two possible sources. First, a pair of nodes (x_i and x_j) which if assigned to *different* partitions in a minimum cut will incur a cost based on the relevant entry in $\widehat{G}$ or $\widetilde{G}$ – corresponds to $\mathbf{G}(\mathbf{x})$ in (16). Second, a pair of nodes where both x_i and x_j is equal to 1 will incur a cost $\frac{2\alpha}{n^2}$ – this corresponds to $\mathbf{R}(\mathbf{x})$ in (16).

Therefore, the definition of *pairwise* costs include the following two scenarios:

$$\Phi_{ij} = \begin{cases} \widehat{G}_{ij} \text{ or } \widetilde{G}_{ij} & \text{if } x_i \neq x_j \\ \frac{2\alpha}{n^2} & \text{if } x_i = x_j = 1 \end{cases} \tag{17}$$

Graph Construction. With the reparameterization given as $\mathbf{\Phi} = [\Phi_j \quad \Phi_{ij}]^T$ done, we can now mechanically follow the recipe in [26,29] to construct a graph $\mathcal{G} = \{V_\mathcal{G}, E_\mathcal{G}\}$ (briefly summarized below). For each variable x_i, we introduce two nodes, ρ_i and $\bar{\rho}_i$ in $V_\mathcal{G}$ (corresponding to literals y_i and $\bar{y}_i$). Hence, the size of the graph is $|V_\mathcal{G}| = 2n$. We also have two special nodes s and t which denote the source and sink resp. We connect each node in $V_\mathcal{G}$ to the source or the sink based on the unary costs, assuming that the source (and sink) partitions correspond to 1 (and 0). The source is connected to the node ρ_i with weight, $-\frac{1\alpha}{n}$ whereas $\bar{\rho}_i$ is connected to the sink with the same weight.

Edges between node pairs in $V_{\mathcal{G}}$ (except source and sink) give pairwise terms of the energy. These edge weights, also given in Table 1, give all possible relationships of pairwise nodes required.

Table 1. Illustration of edge weights introduced in the graph $\mathcal{G}$

Type of edges in $E_{\mathcal{G}}$	Weight of edge
$(\rho_i \to \rho_j), (\bar{\rho_j} \to \bar{\rho_i})$	$\frac{1}{2}\widehat{G}_{ij}$ or $\frac{1}{2}\widetilde{G}_{ij}$
$(\rho_j \to \rho_i), (\bar{\rho_i} \to \bar{\rho_j})$	$\frac{1}{2}\widehat{G}_{ij}$ or $\frac{1}{2}\widetilde{G}_{ij}$
$(\bar{\rho_j} \to \rho_i), (\bar{\rho_i} \to \rho_j)$	$\frac{2\alpha}{n^2}$

A maximum flow/minimum cut procedure on this graph provides a solution to our problem. After the cut is obtained, each node is connected either to the source set or to the sink set. We can obtain a final solution (i.e., 0 or 1 assignment) as,

$$x_j = \begin{cases} 0 \text{ if } v_j \in s, \bar{v}_j \in t \\ 1 \text{ if } v_j \in t, \bar{v}_j \in s \\ \frac{1}{2} \text{ otherwise} \end{cases} \tag{18}$$

The $\frac{1}{2}$ values are then rounded to $\{0, 1\}$.

5.3 Parameter Selection

We now discuss how to choose a suitable α to ensure that the solution to Problem (16) or its associated minimum cut problem is well-balanced. Recall that for every fixed α, Problem (16) is polynomial time solvable. Given an α, let $\mathbf{x}^*(\alpha)$ denote the optimal solution of Problem (16), and let $\phi(\alpha) = \mathbf{R}(\mathbf{x}^*(\alpha))$ be the function value of $\mathbf{R}(\mathbf{x})$ at the optimal solution. We have

Theorem 2. *The function $\phi(\alpha)$ is decreasing in the sense*

$$(\alpha_1 - \alpha_2)(\phi(\alpha_1) - \phi(\alpha_2)) \leq 0, \quad \forall \alpha_1 \neq \alpha_2.$$

Proof. From optimality conditions of Problem (16) we have

$$\mathbf{G}(\mathbf{x}^*(\alpha_1)) + \alpha_1 \mathbf{R}(\mathbf{x}^*(\alpha_1)) \leq \mathbf{G}(\mathbf{x}^*(\alpha_2)) + \alpha_1 \mathbf{R}(\mathbf{x}^*(\alpha_2));$$
$$\mathbf{G}(\mathbf{x}^*(\alpha_2)) + \alpha_2 \mathbf{R}(\mathbf{x}^*(\alpha_2)) \leq \mathbf{G}(\mathbf{x}^*(\alpha_1)) + \alpha_2 \mathbf{R}(\mathbf{x}^*(\alpha_1)).$$

Adding the above two inequalities together, we obtain

$$\alpha_1 \mathbf{R}(\mathbf{x}^*(\alpha_1)) + \alpha_2 \mathbf{R}(\mathbf{x}^*(\alpha_2)) \leq \alpha_1 \mathbf{R}(\mathbf{x}^*(\alpha_2)) + \alpha_2 \mathbf{R}(\mathbf{x}^*(\alpha_1));$$

which implies

$$(\alpha_1 - \alpha_2)(\mathbf{R}(\mathbf{x}^*(\alpha_1)) - \mathbf{R}(\mathbf{x}^*(\alpha_2))) = (\alpha_1 - \alpha_2)(\phi(\alpha_1) - \phi(\alpha_2)) \leq 0.$$

This completes the proof.

The above theorem is interesting in that it suggests that a higher value of α can only improve the balance criteria $\mathbf{R}(\mathbf{x})$. For sufficiently large penalty parameter $\alpha > 0$, the second term $\alpha \mathbf{R}(\mathbf{x})$ in the objective dominates $\mathbf{G}(x)$. Therefore, we can expect that for sufficiently large α, the optimal solution $x^*(\alpha)$ of (16) will be very close to the optimal solution (denoted by $\mathbf{x}^*$) of the following problem $\min \mathbf{R}(\mathbf{x})$ which has a perfect balance. However, for computational purposes, we do not continue optimizing the parameter value α, but are only interested in finding a suitable parameter α such that the solution of (16) meets a pre-specified balance requirement $\mathcal{N}$. This leads us to the simple algorithm in Algorithm 1. The overall algorithm is given in Algorithm 2.

Algorithm 1. Algorithm for $\mathcal{N}$ Balanced Min-Cut

1: **Input:** Graph $\mathcal{G}$, and balance threshold $\mathcal{N}$;
2: Set $k = 0$ and choose a large enough parameter $\alpha_k > 0$;
3: Solve problem (16) with $\alpha = \alpha_k$ and obtain optimal solution $\mathbf{x}(\alpha_k)$;
4: If $R(\mathbf{x}(\alpha_k)) \leq \mathcal{N}$, stop, Output $\mathbf{x}(\alpha_k)$ as final solution.
Else update $k = k + 1, \alpha_k = 2\alpha_{k-1}$, and go to Step 3.

Algorithm 2. Mincut Hashing

1: **Input:** Similarity matrix A, number of bits d, α
2: Initialize X
3: **for** each node i in $[1, d]$ **do**
4: Set up $\bar{A}$, G and $\hat{G}$ (or $\tilde{G}$) according to Sec. 3 or 4.
5: $\mathbf{G} = \hat{G}$ (or $\mathbf{G} = \tilde{G}$)
6: $\hat{x} = \text{argmin}_x \mathbf{G}(\mathbf{x}) + \alpha \mathbf{R}(\mathbf{x})$ solved as a QPBO (using Alg. 1)
7: $X(:, i) = \hat{x}$
8: **end for**

5.4 Out of Sample Extensions

When evaluating the hashing scheme on unseen data, we generate d linear classifiers, using the d bits of training code as labels; these classifiers are then used to predict the hash code for the test example. In a departure from existing methods (except [13]), where the separating hyperplanes are learned within the process of hashing itself, our method solves for the code first and then generates the hyperplanes, which are used to determine the codes for unseen test points. Hyperplanes generated in this manner are based on the maximum margin principle, which offers advantages when generating the hash bits for unseen data.

5.5 Time Complexity

The main step in each iteration is to solve the QPBO on a graph with $O(n)$ nodes. This is repeated d times, once for each bit. If the QPBO is solved using a standard algorithm for min-cut/max-flow, the running time is $O(n|E|)$, where E is the number of edges in the graph. However, we use an implementation of QPBO by [29], which was shown to about 700 times faster than other implementations of QPBO [26], even though its worst case time complexity is inferior ($O(n^2|E||C|)$), where $|C|$ is the largest absolute edge capacity in the network. Such implementations of QPBO are routinely used to solve higher order MRF formulations of image segmentation, where the number of nodes in the graphs (number of pixels) are 10^5 or more.

6 Experiments

Baselines and Datasets. We performed a number of experiments to evaluate the efficacy of our proposal, with both loss functions. We compare our methods to

nine other approaches for binary hashing, including Locality Sensitive Hashing (LSH) [7], KLSH [11], SH [3], MDSH [12], BRE [4], ITQ [5], Anchor Graph Hashing (1, 2 layer) [13] and Fast Hash [16]. We leave out comparison with DGH [8], since it solves the same problem of partitioning the Laplacian of the Anchor Graph as [13], but uses a different optimization technique. Also, note that Fast Hash can be run both in supervised/unsupervised mode, but since all our comparable methods are unsupervised, we compare only with the unsupervised version. We evaluate the algorithms on a number of machine learning and vision datasets, which vary in size, dimensionality, and number of classes. These include toy datasets where the optimal code is known as well as many other datasets like Nursery, MNIST, Caltech101, LabelMe, PhotoTourism and CIFAR.

Parameters. The distance matrix A is set up as the Euclidean distance between features extracted for pair-wise items in each dataset. For image datasets like CIFAR and Phototourism, we extract HOG features for each image. For machine learning datasets, we use the features provided in the dataset itself. Also, α is set to 0.1 throughout. An initial value of the solution is generated using random hyperplanes similar to LSH.

Design. Broadly, we evaluate two aspects: (i) how well do the set of obtained hash-codes *approximate the original distances*, and (ii) how well do the obtained hash codes *preserve semantic labeling* of the examples. To address the issue of *distance approximation* in (i), we use a two-fold approach: (a) We construct toy datasets, where the ground-truth code (and the optimal objective) is known. So, how well do the eleven algorithms optimize the objective relative to the optimal? (b) We evaluate the goodness of nearest neighbor search in the binary/embedded space. That is, precision of how many neighbors returned by our hash code are "true" neighbors? Next, note that we can quantitatively evaluate the issue in (ii) above because many datasets here have semantic labels for examples, so data points with the same 'labels' are considered "semantic" neighbors as well. For *semantic label experiments*, the data is divided into *training* set which is used to construct the distance matrix on which codes are learned and *test data*, which is used for evaluation only. For datasets where labels are available, we report accuracies of the k nearest Neighbors ($k = 3$) of a given query, w.r.t. the same class labels. This will help us demonstrate whether semantic concepts can be identified using such an approach. These experiments are described next.

6.1 Distance Approximation

Minimization of Objective: We simulate toy datasets where the optimal code is known. To do this, we randomly generate a code(X), and then create a distance matrix A, which is a close estimate (modulo a small error) of the Hamming Distance (H) obtained from X. We repeat this experiment by varying n, d and the magnitude of noise added to the Hamming distance matrix (to obtain A). In some sense, this represents the *ideal* input to the problem (though we may still have multiple binary codes for the same distance matrix). For other comparable methods, which require the affinity matrices, we generate that using the identity

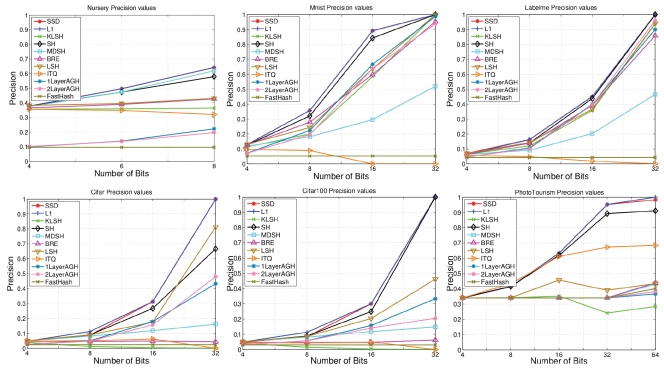

Fig. 1. Precision Values of Nursery, Mnist, Labelme, Cifar, Cifar 100 and Phototourism

Table 2. Ranking results wrt to SSD and ℓ_1 objectives.

Alg on Obj	Rank 1	Rank 2	Rank 3
Alg-ℓ_1 on SSD	85 %	15 %	0
Alg-ℓ_1 on ℓ_1	97 %	3 %	0
Alg-SSD on SSD	60 %	36 %	4 %
Alg-SSD on ℓ_1	57 %	40 %	3 %

$A_{\text{aff}} = \frac{1}{2}(2d\mathbf{1}_{n\times n} - A)$. We run all 11 methods on each of the settings and record rankings of the algorithms in terms of decreasing the objective, for both loss functions. For convenience, below, we will call our actual algorithms Alg-SSD and Alg-ℓ_1 and the distance functions as SSD and ℓ_1 respectively.

Table 2 shows the percentage of times (averaged across 20 realizations), Alg-SSD and Alg-ℓ_1 achieves ranks 1 to 3 (1 being the best, ranks are only shown up to 3 as neither algorithm does worse than third place in any iteration), while minimizing these objectives. The plot shows that Alg-ℓ_1 is the best, minimizing the ℓ_1 objective better than all other algorithms, with a mean weighted rank of 1.1. Alg-SSD is a close second, with a weighted rank of 1.45 across all the runs, compared to other algorithms. These results show that our algorithms do much better than the alternatives in getting close to the global optima.

Generalization to Unseen Data: Instead of looking at faithfully approximating all pair-wise distances, one may attempt to approximate distances of only the nearest neighbors to a query point. To do so, we adopt the following procedure. We divide the data into training and test sets (randomly selected, 1000 points each), and compute distances between training to training and training to test datasets. In order to evaluate how well NN distances are estimated using the hash codes on unseen data, we first define a threshold, such that if the original distance of a pair of points is less than the threshold, they are considered "true neighbors". Given a query and a threshold on the Hamming distance (set to

3), the retrieved items are all examples whose Hamming distance to the query is below this threshold. We compute *precision* as the proportion of retrieved points that are indeed true neighbors. Figure 1 shows the precision as a function of the number of bits for various datasets. In general, our method performs well compared to the baselines.

6.2 Semantic Labels

Datasets such as Nursery, Mnist, Caltech 101 and Cifar (both datasets) have a semantic label associated with each example/image. We present results on these datasets next.

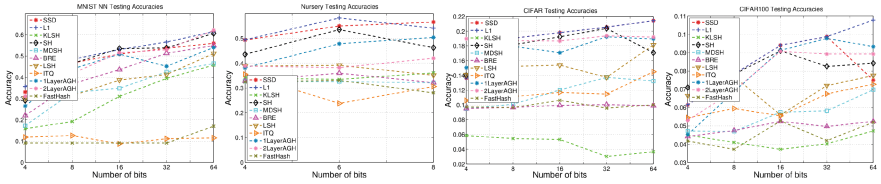

Fig. 2. Semantic Label NN Accuracy for Mnist, Nursery and Cifar, Cifar 100

Generalization to Unseen Data as a Function of Labels: We show several plots of accuracy of k-nearest neighbors having the same label as the query (Fig. 2) for Nursery, Mnist, Cifar and Cifar100 using binary hash codes obtained using a (randomly selected) training set of 2000 and generalized to a much larger testing set (which is typically the rest of the data). Note that some of the datasets (such as Cifar) are harder to classify, therefore the relative percentage accuracy varies depending on the dataset. However, in almost all cases, our models show impressive performance in finding neighbors which have the same class label as the query point.

Effect of Number of Classes in Semantic Label Experiments: We used Caltech101 to evaluate how the number of semantic class labels affect the performance of our algorithm, since the dataset has a large number of classes (up to 101) and a small number of data items per class. Here, we use (the mean of a small set of) pre-computed kernel matrices obtained from UCSD MKL dataset since most of the comparable methods can be run on the kernel matrices (which can be used to define similarities also). Whenever actual features were needed for an algorithm, we generated them by implementing the approach in [30]. However, since the features for the test dataset is unavailable, we limit our evaluation to the training dataset only. For the same reason, Fast Hash (which needs features

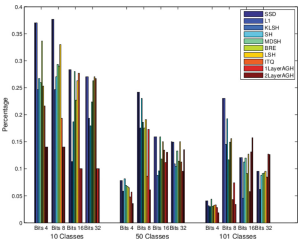

Fig. 3. UCSD accuracy results.

explicitly) could not be applied on this dataset. Figure 3 shows the results on the Caltech 101 dataset, using 10, 50 and 101 classes. The plot shows that in most settings, our methods outperform the other baselines. In addition, increasing the number of classes does affect performance but not drastically.

Running Time: Each iteration of QPBO takes about one second and this is repeated d (number of bits) times. So, given a choice of number of bits, the run time varies from 4 secs to about a minute (when the number of bits is 64). This is comparable to most of the methods tested in this paper, except Spectral Hashing (and MDSH), which do not depend on the number of bits, since their main computational bottleneck is computing the eigen vectors of a matrix once.

7 Discussion

Here we briefly discuss some issues related to our algorithm.

Size of Training Data. We performed limited experiments to see if increasing the size of the training set improves generalization performance but found that in general, the relative improvement saturates pretty quickly. Therefore, a moderate size training set (if chosen randomly) is enough to ensure good generalization. Similar training sizes have also been used in [11].

Rounding of Half Integral Variables in QPBO. In our case, we randomly round the $\frac{1}{2}$ variables to zero or one. Generally, this happens to fewer than 20 % of the variables, so it works fairly well in practice. If in the worst case, a large number of variables have half-integral solutions, there is a mature body of work to deal with this effectively. Roof duality [29,31,32] are specifically addresses this issue and can be used though we did not find it necessary.

Existing Literature. In this paper, we discuss 17 other works on binary hashing which are closely related to the ideas presented here. A more comprehensive (but by no means, exhaustive) list can be found in the survey [33].

8 Conclusion

We present a fully combinatorial algorithm for learning hash codes whose Hamming distance closely approximates a target similarity matrix. In a landscape dominated by continous optimization methods for binary hashing, our algorithms provide a different view point — a maximum flow/minimum cut based method in the unsupervised setting. The implementation is simple and the solutions for each hash code bit is provably partially-optimal. Experimentally, we show that these methods exhibit competitive performance, compared against nine state of the methods on seven different datasets.

Acknowledgments. This research is funded by NIH R01 AG040396, NSF CAREER 1252725, NSF CGV 1219016 and NSF CMMI 1359548 and 1537712.

References

1. Mukherjee, L., Singh, V., Peng, J.: Scale invariant cosegmentation for image groups. In: CVPR (2011)
2. Mukherjee, L., Singh, V., Dyer, C.R.: Half-integrality based algorithms for cosegmentation of images. In: CVPR (2009)
3. Weiss, Y., Torralba, A., Fergus, R.: Spectral hashing. In: NIPS (2008)
4. Kulis, B., Darrell, T.: Learning to hash with binary reconstructive embeddings. In: NIPS (2009)
5. Gong, Y., Lazebnik, S.: Iterative quantization: a procrustean approach to learning binary codes. In: CVPR (2011)
6. Norouzi, M., Fleet, D.M.: Minimal loss hashing for compact binary codes. In: Proceedings of the 28th International Conference on Machine Learning (ICML-11) (2011)
7. Indyk, P., Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality. In: Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing, STOC 1998, pp. 604–613. ACM, New York (1998)
8. Liu, W., Mu, C., Kumar, S., Chang, S.: Discrete graph hashing. In: NIPS (2014)
9. Salakhutdinov, R., Hinton, G.: Semantic hashing. Int. J. Approximate Reasoning **50**(7), 969–978 (2009)
10. Krizhevsky, A., Hinton, G.E.: Using very deep autoencoders for content-based image retrieval. In: ESANN (2011)
11. Kulis, B., Grauman, K.: Kernelized locality-sensitive hashing for scalable image search. In: 2009 IEEE 12th International Conference on Computer Vision, pp. 2130–2137. IEEE (2009)
12. Weiss, Y., Fergus, R., Torralba, A.: Multidimensional spectral hashing. In: Fitzgibbon, A., Lazebnik, S., Perona, P., Sato, Y., Schmid, C. (eds.) ECCV 2012. LNCS, vol. 7576, pp. 340–353. Springer, Heidelberg (2012). doi:10.1007/978-3-642-33715-4_25
13. Liu, W., Wang, J., Kumar, S., Chang, S.: Hashing with graphs. In: ICML (2011)
14. Mukherjee, L., Ravi, S.N., Ithapu, V.K., Holmes, T., Singh, V.: An NMF perspective on binary hashing. In: IEEE International Conference on Computer Vision (ICCV), pp. 4184–4192 (2015)
15. Zhang, D., Wang, J., Cai, D., Lu, J.: Self-taught hashing for fast similarity search. In: SIGIR, pp. 18–25 (2010)
16. Lin, G., Shen, C., Suter, D., van den Hengel, A.: A general two-step approach to learning-based hashing. In: ICCV (2013)
17. Shakhnarovich, G., Viola, P., Darrell, T.: Fast pose estimation with parameter-sensitive hashing. In: ICCV (2003)
18. Neyshabur, B., Srebro, N., Salakhutdinov, R., Makarychev, Y., Yadollahpour, P.: The power of asymmetry in binary hashing. In: NIPS (2013)
19. Ge, T., He, K., Sun, J.: Graph cuts for supervised binary coding. In: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (eds.) ECCV 2014. LNCS, vol. 8695, pp. 250–264. Springer, Heidelberg (2014). doi:10.1007/978-3-319-10584-0_17
20. Shen, F., Shen, C., Liu, W., Shen, H.T.: Supervised discrete hashing. In: CVPR (2015)
21. Li, H., Liu, W., Ji, H.: Two-stage hashing for fast document retrieval. In: ACL (2014)
22. Stoer, M., Wagner, F.: A simple min-cut algorithm. J. ACM **44**(4), 585–591 (1997)

23. Wu, Z., Leahy, R.: An optimal graph theoretic approach to data clustering: theory and its application to image segmentation. IEEE Trans. Pattern Anal. Mach. Intell. **15**(11), 1101–1113 (1993)
24. Andreev, K., Racke, H.: Balanced graph partitioning. Theory Comput. Syst. **39**(6), 929–939 (2006)
25. Chen, Y., Zhang, Y., Ji, X.: Size regularized cut for data clustering. In: Advances in Neural Information Processing Systems, pp. 211–218 (2005)
26. Boros, E., Hammer, P.L.: Pseudo-Boolean optimization. Discrete Appl. Math. **123**(1–3), 155–225 (2002)
27. Kolmogorov, V., Rother, C.: Minimizing nonsubmodular functions with graph cuts-a review. IEEE Trans. Pattern Anal. Mach. Intell. **29**(7), 1274–1279 (2007)
28. Rother, C., Kohli, P., Feng, W., Jia, J.: Minimizing sparse higher order energy functions of discrete variables. In: IEEE Conference on Computer Vision and Pattern Recognition, 2009, CVPR 2009, pp. 1382–1389. IEEE (2009)
29. Rother, C., Kolmogorov, V., Lempitsky, V., Szummer, M.: Optimizing binary MRFs via extended roof duality. In: IEEE Conference on Computer Vision and Pattern Recognition, 2007, CVPR 2007, pp. 1–8. IEEE (2007)
30. Balcan, M.F., Blum, A., Vempala, S.: Kernels as features: on kernels, margins, and low-dimensional mappings. Mach. Learn. **65**(1), 79–94 (2006)
31. Kahl, F., Strandmark, P.: Generalized roof duality for pseudo-boolean optimization. In: 2011 IEEE International Conference on Computer Vision (ICCV), pp. 255–262. IEEE (2011)
32. Kolmogorov, V.: Generalized roof duality and bisubmodular functions. In: Advances in Neural Information Processing Systems, pp. 1144–1152 (2010)
33. Wang, J., Shen, H.T., Zhang, T.: A survey on learning to hash. MSRC Technical Report (2014)