Non-transitive Hashing with Latent Similarity Components

Mingdong Ou[1], Peng Cui[1], Fei Wang[2], Jun Wang[3], Wenwu Zhu[1]
[1]Tsinghua National Laboratory for Information Science and Technology
Department of Computer Science and Technology, Tsinghua University. Beijing, China
[2]Department of Computer Science and Engineering, School of Engineering, University of Connecticut. Storrs, U.S.
[3]Data Science, Alibaba Group. Seattle, WA, U.S.
oumingdong@gmail.com,cuip@tsinghua.edu.cn, fei_wang@engr.uconn.edu,
jwang@ee.columbia.edu, wwzhu@tsinghua.edu.cn

ABSTRACT

Approximating the semantic similarity between entities in the learned Hamming space is the key for supervised hashing techniques. The semantic similarities between entities are often non-transitive since they could share different latent similarity components. For example, in social networks, we connect with people for various reasons, such as sharing common interests, working in the same company, being alumni and so on. Obviously, these social connections are non-transitive if people are connected due to different reasons. However, existing supervised hashing methods treat the pairwise similarity relationships in a simple and unified way and project data into a single Hamming space, while neglecting that the non-transitive property cannot be adequately captured by a single Hamming space. In this paper, we propose a non-transitive hashing method, namely *Multi-Component Hashing* (*MuCH*), to identify the latent similarity components to cope with the non-transitive similarity relationships. *MuCH* generates multiple hash tables with each hash table corresponding to a similarity component, and preserves the non-transitive similarities in different hash table respectively. Moreover, we propose a similarity measure, called *Multi-Component Similarity*, aggregating Hamming similarities in multiple hash tables to capture the non-transitive property of semantic similarity. We conduct extensive experiments on one synthetic dataset and two public real-world datasets (i.e. DBLP and NUS-WIDE). The results clearly demonstrate that the proposed *MuCH* method significantly outperforms the state-of-art hashing methods especially on search efficiency.

Categories and Subject Descriptors

H.4 [**Information System Applications**]: Miscellaneous

Keywords

Non-transitive similarity; Hashing; Similarity components

KDD'15, August 10-13, 2015, Sydney, NSW, Australia.

ACM 978-1-4503-3664-2/15/08 ...$15.00.
DOI: http://dx.doi.org/10.1145/2783258.2783283.

1. INTRODUCTION

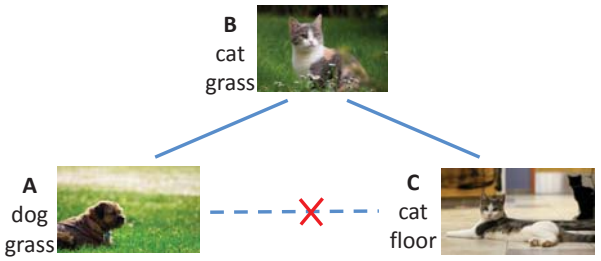


Figure 1: Non-transitive image triangle (i.e. (A, B, C)). Although (A, B) and (B, C) are *similar* pairs, A and C are *dissimilar*, which violates the transitive intuition on similarity. This is because the *similar* pairs ((A, B) and (B, C)) are determined by different similarity components, i.e. object (cat or dog) or scene (grass or floor). The blue full lines represent *similar* relationship, and the blue dashed lines with red cross represent *dissimilar* relationship.

With the explosive growth of data, similarity search is becoming increasingly important for a wide range of large scale applications, including image retrieval [9, 21], document search [30], and recommendation systems [16, 15, 37]. Due to the simplicity and efficiency, hashing techniques are one of the best options for indexing large scale datasets and performing approximate nearest neighbor search. In particular, recent hashing methods often employ machine learning techniques to leverage supervised information like pairwise constraints to design more efficient hash codes to search semantically similar entities [35, 39, 22, 28]. The key for the learning based hashing methods is to approximate the semantic similarity between entities in the learned Hamming space. However, an important but often neglected intuition is that similarity between entities is often non-transitive since the semantic similarity could be determined by different latent components.

As illustrated in Figure 1, image A is similar to image B for a common scene (i.e. grass), while image B is similar with image C for a common object (i.e. cat). However, image A is dissimilar with image C since they share no common components, neither the object nor the scene. Due to the existence of latent similarity components, such a similarity metric has non-transitive triangle relationships, which can be observed in many real world applications. In `Facebook`,

for instance, people are connected by friendship links for various reasons, such as sharing common interests, working in the same industry, being graduated from the same school, and so on. Apparently, such friendship connections could be non-transitive. Although non-transitive similarity exists in many scenarios, we usually can only observe the pairwise relationship between entities while the underlying similarity components, which is the main cause of non-transitive similarity, are unknown.

In literature, most of the supervised hashing techniques use the pairwise relationship links without identifying the true similarity components, and neglect the non-transitive property of the semantic metric. In general, it is difficult to capture the true non-transitive triangle relationship in a single metric space [7, 33]. As an example illustrated in Figure 2, we have a non-transitive relationship between three images $\{A, B, C\}$. Although the similarities between $\{A, B, C\}$ are determined by different components (scene or object), existing supervised hashing methods treat them in a unified way, e.g. all the similar image pairs arise from the common cause (i.e. similar on both scene and object). Considering that image A is similar to image B and image B is similar to image C, A need to be close to B, and B need to be close to C in the Hamming space learned by existing supervised hashing methods. Thus, A will be relatively close to C, which violates the truth that A is dissimilar to C. Therefore, in order to fully capture the non-transitive similarity while maintaining the retrieval accuracy, we need to identify the latent similarity components and design hash functions upon each similarity component, namely non-transitive hashing.

However, it is very challenging to design an effective non-transitive hashing technique. First, since we can only observe the similarity relationships between entities, it is difficult to identify different latent similarity components explicitly from a single pairwise relationship. Second, the *similar* relationship means a pair of entities share at least one common similarity components, and the *dissimilar* relationship means a pair of entities share no common similarity component. Therefore, it is desired to distinguish these two types of relationships with the identified latent similarity components. Finally, the volume of labeled pairwise similarity is often limited. Hence, the learning algorithm should be able to handle the issue of limited labels, while avoiding overfitting.

In this paper, we propose a novel non-transitive hashing algorithm, *Multi-Component Hashing* (*MuCH* for short), to address the above challenges. *MuCH* jointly learns multiple linear hash functions to project entities into multiple hash tables, and approximates each single latent similarity component using a different hash table to preserve non-transitive properties (see the top part in Figure 2). By aggregating multiple hash tables, we propose a unified similarity measure, called multi-component Hamming similarity, to model the different properties of the *similar* and *dissimilar* relationships. Besides, in order to cope with limited and noisy pairwise labels and avoid overfitting, a regularizer is designed to maximize the information captured by hash codes. Moreover, We optimize the objective by a gradient descent method. The time complexity of each iteration is linear with the size of training set, which is scalable for large-scale data. In the testing scenario, given a query, we assign it a hash code in each hash table, and use them to merge similar entities across different hash tables. After that, an aggregation strategy is designed to aggregate the retrieval results from different hash tables, and a simple metric that matches the strategy is proposed for ranking these results. Extensive experiments are conducted on one synthetic dataset and two public real-world datasets, i.e. NUS-WIDE [8] and DBLP[1]. The results demonstrate that our method, *MuCH*, can effectively identify the similarity components and outperform the other comparative methods, especially on search efficiency.

The rest of this paper is presented as follows. In Section 2, we give a brief review of the related works. In Section 3, we describe the framework and model of *MuCH*. Then, we describe the experiment setting and analyse the results in Section 4. Finally, we conclude our work in Section 5.

2. RELATED WORKS

In this section, we give a brief review of works on two fields, i.e. non-transitive similarity and hashing.

2.1 Non-transitive Similarity Learning

Although most of existing methods [6] assume that similarity are all transitive, non-transitive similarity have been studied in different fields. In metric learning, researchers realize the limitation of metric space on capturing the non-transitive similarity, and propose some effective non-metric learning algorithms for non-transitive similarity [7, 33]. Similarity Component Analysis (SCA) [7] proposes a probabilistic graphical model to discover latent pairwise similarity components, and aggregating them as the final similarity. Note that the similarity components discovered in our work are entity-wise, and each entity is represented by a combination of similarity components. Multiple maps t-SNE [33] aims to represent non-transitive similarity and central objects in two-dimensional visualizations. In social networks, many works [32, 10, 13, 2, 1] focus on extracting the multiple types of relationship or finding the overlapping communities. Mixed membership stochastic blockmodels [2] discovers overlapping communities in networks, and represents vertices in networks with mixed membership to communities. However, these methods cannot work in the scenario of hashing. Though they can effectively measure the non-transitive similarity, they cannot conduct efficient search for similar entities in very large scale.

2.2 Hashing

With the rapid increase of the data volume, hashing is proposed to solve the efficiency problem on approximate nearest neighbors search in large-scale high-dimensional data. Locality Sensitive Hashing (i.e. LSH) methods [14, 4] are first proposed, which generate hash codes with random projections or permutations. Moreover, Mu et. al. [27] apply LSH to non-metric similarity. While locality sensitive hashing is independent with data, and has to generate long hash code to achieve high accuracy, Spectral hashing [38] is proposed to learn hash functions based on the data distribution, and achieves much compact hash code and higher accuracy. Some more unsupervised hashing methods [23, 12, 17, 11] based on data distributions are proposed later. For the well-known semantic gap between low-level features and semantic concepts, the performance of unsupervised hashing suffers bottleneck. Semi-supervised or supervised hashing methods [36, 22, 34, 19, 20] exploit the labeled pairwise simi-

[1] http://www.informatik.uni-trier.de/~ley/db/

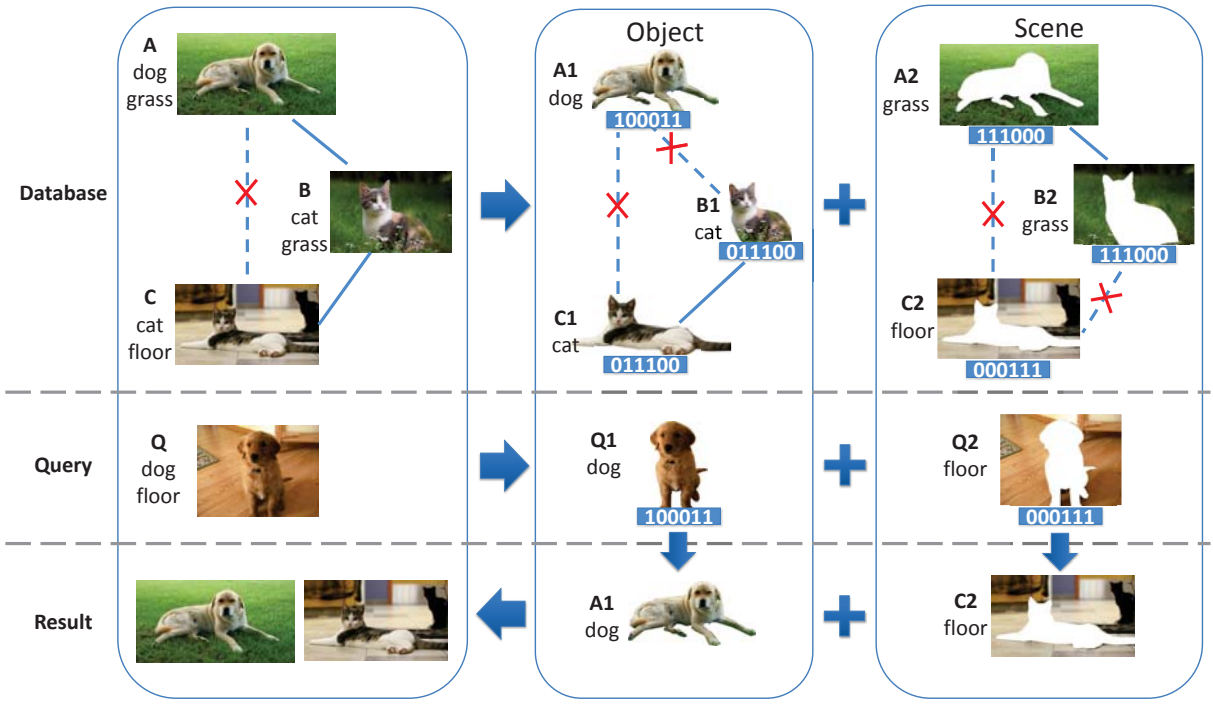


Figure 2: The framework of *Multi-Component Hashing* (*MuCH*). The image triangle (A, B, C) is a non-transitive triangle. The blue full lines between images represent *similar* relationship, and the blue dashed lines with red cross represent *dissimilar* relationship. The text in the left of images are id and label of images (latent in our setting, showed here just for clear description), and the binary codes below the images are their hash codes. The framework splits into three parts by gray dashed lines corresponding to three procedure of *MuCH*, i.e. offline hash function learning, online hash code generation and similar entity retrieval. In the top part, *MuCH* learns multiple hash tables to capture the latent similarity components (i.e. object and scene in above figure), and approximates the non-transitive triangle with multiple inherently transitive triangle in generated hash tables. In the middle part, given a query, say image Q, *MuCH* first generates hash codes in different hash tables, then uses these hash codes to search similar entities (i.e. A1 and C2) in corresponding hash tables. Finally, in the bottom part, *MuCH* aggregates all the results retrieved from multiple hash tables.

larity relationship between entities to capture the high-level semantics. Moreover, some multi-modal hashing methods [5, 18, 41, 31, 40, 29] are proposed to exploit multiple features for hashing to get higher accuracy. However, these hashing methods cannot discover the latent similarity components and capture the non-transitive similarity.

Another class of hashing methods that is related to our work is the hashing methods using multiple hash tables. In order to improve the recall of hashing and preserve the precision at the same time, some multi-table hashing methods [39, 24] are proposed, where complementary hashing [39] learns multiple hash tables with boosting methods, and reciprocal hashing [24] learns multiple hash tables by selecting hash functions from a pool of hash functions that learned by many hashing methods. Since these methods treat similarity and dissimilarity relationships in the same way, these multi-table hashing methods are not designed to identify latent similarity components and cannot capture non-transitive similarity. Hashing on multi-label data [26, 25] learn different hash tables for different known labels. Heterogeneous hashing [28] generates a variant of original hash table in each domain to search, in order to capture the specific characteristics of target domain. The setting of the above two class of hashing methods are different from our setting where similarity components (i.e. label or domain) are latent. So, how to identify latent similarity components and capture non-transitive similarity in hashing remains an open problem.

3. MULTI-COMPONENT HASHING

In this section, *Multi-component Hashing* (*MuCH*) is proposed to capture the non-transitive property of semantic similarity. Thus, we can perform efficient and accurate similarity search on data where similarity is determined by multiple similarity components. In the remaining of this section, we will first give a brief overview of the framework showed in Figure 2. Then, we will formally formulate the problem and propose our model. Finally, an efficient optimization algorithm will be given along with theoretical complexity analysis.

3.1 Framework

Figure 2 shows the framework of *MuCH*. *MuCH* consists of two phase: offline learning (the top part of Figure 2) and online retrieval (the bottom two parts of Figure 2). For simplicity, we just show a case with two hash tables. It is easy to generalize to more hash tables. In the top part of Figure 2, *MuCH* learns two linear projections that project entities in original feature space into two hash tables, and

each hash table corresponds to a similarity component (object or scene). For simplicity, we call the two hash tables object table and scene table respectively. The hash codes of *similar* pair (B, C) are close in the object table, and the hash codes of *similar* pair (A, B) are close in the scene table. Meanwhile, the hash codes of *dissimilar* pair (A, C) are far from each other in both hash tables. Thus, because of the inherent transitivity property of hash tables, the similar pair (A, B) cannot be well preserved in object table and (B, C) cannot be well preserved in scene table. But, if we apply the combination rule that a pair of entities are similar when their hash codes are close in at least one hash table and a pair of entities are dissimilar when their hash codes are far from each other in all the hash tables, we can reconstruct the non-transitive triangle (A, B, C) by combining the corresponding triangles (inherently transitive) in two hash tables.

In the bottom part of Figure 2, given a query image Q, *MuCH* generates hash codes for Q in two hash tables respectively, then uses these hash codes to search for similar images in corresponding hash tables, we can find that Q is similar to A in object table and similar to C in scene table. Finally, we aggregate the similar images (i.e. A and C) retrieved from different hash tables as the search results for Q.

3.2 Notations

Suppose there are N entities with feature matrix $X = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N] \in \mathbb{R}^{L \times N}$, where L is dimensionality of feature. M hash tables are generated by a set of linear projection matrices $\mathbf{W} = [\mathbf{W}^1, \mathbf{W}^2, \cdots, \mathbf{W}^M] \in \mathbb{R}^{L \times KM}$, where $\mathbf{W}^m \in \mathbb{R}^{L \times K}$ and K is the length of a hash code (i.e. the number of hash bits) in each hash table. Then, we calculate m-th hash table $\mathbf{H}^m \in \{-1, 1\}^{K \times N}$ by

$$\mathbf{H}^m = sgn(\mathbf{W}^{m\top}\mathbf{X}) \tag{1}$$

where

$$sgn(x) = \begin{cases} 1 & x > 0 \\ -1 & x \leq 0 \end{cases}$$

$\{\mathbf{H}^m\}$ are concatenated as $\mathbf{H} = [\mathbf{H}^{1\top}, \mathbf{H}^{2\top}, \cdots, \mathbf{H}^{M\top}]^\top$.

For simplicity, we give a unified definition for variables with subscript. For a matrix, say $\mathbf{D}$, D_{ij} denotes the element in i-th row and j-th column and $\mathbf{d}_j$ denotes the j-th column.

We use the matrix $\mathbf{R} \in \{-1, 0, 1\}^{N \times N}$ to formulate the observed pairwise relationships, where 1 means similar, -1 means dissimilar and 0 means unobserved. We denote the number of non-zero elements in $\mathbf{R}$ as N_e, i.e. the number of observed relationships between entities. Let $\mathcal{M} = \{(i, j)|R_{ij} = 1\}$ be the set of *similar* pairs, and $\mathcal{C} = \{(i, j)|R_{ij} = -1\}$ be the set of *dissimilar* pairs. The similarity between hash codes in m-th hash table is denoted as $\mathbf{S}^m \in [0, 1]^{N \times N}$, whose elements is defined as

$$S_{ij}^m = \mathbf{h}_i^\top \mathbf{h}_j$$

Thus, $S_{ij}^m \in [-K, K]$.

3.3 Problem Formulation

We need to learn the linear projection matrix $\mathbf{W}$ so that the similarity relationship matrix $\mathbf{R}$ can be approximated by the similarity matrices $\{\mathbf{S}^m\}_{m=1}^M$. First, we define the aggregated similarity as

$$S_{ij} = g(S_{ij}^1, S_{ij}^2, \cdots, S_{ij}^M) \in [-K, K] \tag{2}$$

Thus, we denote the aggregated similarity matrix as $\mathbf{S} = \{S_{ij}\}^{N \times N}$. As R_{ij} is binary, the approximation problem can be formulated as restricting that S_{ij} and R_{ij} have common sign, formally

$$\max_{\mathbf{W}} \sum_{i,j} sgn(R_{ij} S_{ij}) \tag{3}$$

As the sgn function is not continuous, the optimization problem are often intractable. So, we relax the above problem as below

$$\min_{\mathbf{W}} \sum_{i,j} f(R_{ij} S_{ij}) \tag{4}$$

where function $f(x)$ is continuous and monotonously decreasing with respect to x. Thus, S_{ij} should be large when $R_{ij} = 1$, and S_{ij} should be small when $R_{ij} = -1$.

3.4 Model Formulation

In this section, we focus on designing the function expressions in formula (4) (i.e. f and g) to well capture the properties of data, such as non-transitive similarity and sparsity.

3.4.1 Multi-Component Similarity

The design of aggregated similarity function g is the key of our model. To approximate $\mathbf{R}$, it needs to capture the non-transitive property of semantic similarities. According to the multiple similarity components assumption that explains the non-transitive phenomena, we need to preserve at least one of the similarities $\{S_{ij}^m\}_{m=1}^M$ large when $R_{ij} = 1$, and all of the similarities $\{S_{ij}^m\}_{m=1}^M$ small when $R_{ij} = -1$. It is simply equivalent to that the maximum similarity $max\{S_{ij}^m\}_{m=1}^M$ should be large when $R_{ij} = 1$ and small when $R_{ij} = -1$. Formally, we give the lemma below

LEMMA 1. *Given a constant s_c, $\exists s \in \{s^m\}$ so that $s \geq s_c$ if and only if $\max\{s^m\} \geq s_c$; $\forall s \in \{s^m\}$, $s \leq s_c$ if and only if $\max\{s^m\} \leq s_c$.*

According to Lemma 1, we just need to work on $\max\{S_{ij}^m\}_{m=1}^M$ to capture the non-transitive similarities. Moreover, we can find that $\max\{S_{ij}^m\}_{m=1}^M$ also matches the request to aggregated similarity S_{ij} in formula (4), so we can define the aggregated similarity function as

$$g(S_{ij}^1, S_{ij}^2, \cdots, S_{ij}^M) = \max\{S_{ij}^m\}_{m=1}^M \tag{5}$$

However, with the maximum function, the minimization problem in formula (4) will be hard to solve. So, we use softmax function to approximate the maximum function as below

$$\max\{S_{ij}^m\}_{m=1}^M \approx \frac{\sum_{m=1}^M S_{ij}^m e^{S_{ij}^m}}{\sum_{m=1}^M e^{S_{ij}^m}} \tag{6}$$

Finally, we define the final aggregated similarity S_{ij}, called *Multi-Component Similarity*, based on softmax as below

$$S_{ij} = \frac{\sum_{m=1}^M S_{ij}^m e^{S_{ij}^m}}{\sum_{m=1}^M e^{S_{ij}^m}} \tag{7}$$

3.4.2 Final Objective

With the aggregated similarity defined, we can define how well the aggregated similarity S_{ij} approximates semantic similarity R_{ij}, i.e. loss function f. According to formula (4), loss function $f(x)$ should be continuous and monotonously decreasing with respect to x. Moreover, as $\mathbf{R}$ is often very sparse, the loss function should lead to model with strong generalization ability. So, we adopt the logistic loss, which can effectively avoid overfitting, as below:

$$f(x) = log(1 + e^{-x}) \tag{8}$$

We will explain the advantage of logistic loss in detail in section 3.5 where the explanation based on gradient will be more intuitive.

Thus, by substituting Equation (8) into Formula (4), we can get the empirical loss function of the whole data:

$$\mathcal{LE} = \sum_{i,j} log(1 + e^{-R_{ij}S_{ij}}). \tag{9}$$

To alleviate the overfitting problem further, we restrict that the linear projections should be little correlated with each other, thus we can preserve more information in hash codes [35]. Moreover, to avoid trivial results, we restrict that the magnitude of linear projections should be small. So, we add two regularizers to the final objective, that is

$$\mathcal{LR} = \gamma_1 \sum_{i \neq j} (\mathbf{w}_i^\top \mathbf{w}_j)^2 + \gamma_2 \|\mathbf{W}\|_{\mathcal{F}}^2 \tag{10}$$

where γ_1 and γ_2 are two constant parameters to tune the contribution weight of the regularizers.

Then, the final objective that combines empirical loss function and regularizers is

$$\min_{\mathbf{W}} \quad \mathcal{L} = \mathcal{LE} + \mathcal{LR} \tag{11}$$

where

$$\mathcal{LE} = \sum_{i,j} log(1 + e^{-R_{ij}S_{ij}}) \tag{12}$$

$$\mathcal{LR} = \gamma_1 \sum_{i \neq j} (\mathbf{w}_i^\top \mathbf{w}_j)^2 + \gamma_2 \|\mathbf{W}\|_{\mathcal{F}}^2 \tag{13}$$

3.5 Optimization

It is hard to get an analytic solution of the final objective (11), we solve it with iterative optimization algorithm.

However, the hash function (Equation (1)) is not continuous which make the final objective intractable. So, we approximate the sign function with the smooth sigmoid function [22] as below

$$\mathbf{H} = \frac{2}{1 + e^{-\mathbf{W}^\top \mathbf{X}}} - 1 \tag{14}$$

Thus, the final objective is differentiable.

We optimize the final objective with *Block Coordinate Descent* (BCD), and the step of Gradient Descent is determined by line search (see Algorithm 1). The gradient of $\mathcal{L}$ is

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \frac{\partial \mathcal{LE}}{\partial \mathbf{W}} + \frac{\partial \mathcal{LR}}{\partial \mathbf{W}} \tag{15}$$

Algorithm 1 Multi-Component Hashing (MuCH)

Require: feature matrix of training set $\mathbf{X}$, adjacency matrix $\mathbf{R}$, number of hash bits K, number of hash tables M

Ensure: hash codes $\mathbf{H}$, linear hash functions $\mathbf{W} = [\mathbf{W}^1, \mathbf{W}^2, \cdots, \mathbf{W}^M]$

```
initialize W by PCA
while the value of objective function don't converge do
    for all linear projections {W^m} of hash tables do
        calculate the gradient of ∂L/∂W^m
        determine the step a with line search
        update W^m by W^m = W^m − a ∗ ∂L/∂W^m
    end for
end while
encoding H by H = sgn(W^⊤ X)
```

The gradient of empirical objective $\mathcal{LE}$ is

$$\frac{\partial \mathcal{LE}}{\partial \mathbf{w}_k^m} = \sum_{i \neq j} \frac{\partial f(R_{ij}S_{ij})}{\partial \mathbf{w}_k^m}$$

$$\frac{\partial f(R_{ij}S_{ij})}{\partial \mathbf{w}_k^m} = \frac{-R_{ij}}{1 + e^{R_{ij}S_{ij}}} * \frac{\partial S_{ij}}{\partial \mathbf{w}_k^m} \tag{16}$$

$$\frac{\partial S_{ij}}{\partial \mathbf{w}_k^m} = \frac{(1 + S_{ij}^m - S_{ij})e^{S_{ij}^m}}{\sum_{n=1}^{M} e^{S_{ij}^n}} * \frac{\partial S_{ij}^m}{\partial \mathbf{w}_k^m} \tag{17}$$

$$\frac{\partial S_{ij}^m}{\partial \mathbf{w}_k^m} = H_{kj} \frac{\partial H_{ki}^m}{\partial \mathbf{w}_k^m} + H_{ki} \frac{\partial H_{kj}^m}{\partial \mathbf{w}_k^m}$$

$$\frac{\partial H_{ki}^m}{\partial \mathbf{w}_k^m} = \frac{2e^{-\mathbf{w}_k^{m\top} \mathbf{x}_i}}{(1 + e^{-\mathbf{w}_k^{m\top} \mathbf{x}_i})^2} \mathbf{x}_i$$

From Equation (16), we can see that the smaller $R_{ij}S_{ij}$ is, the larger the magnitude of gradient is. As that $R_{ij}S_{ij}$ is small means that R_{ij} has not been approximated well, the optimization algorithm will approximate the poorly approximated R_{ij} with higher priority. Thus, we can expect that all the semantic similarities will be approximated as well as possible, and the model will have strong generalization ability.

The gradient of regularizers $\mathcal{LR}$ is

$$\frac{\partial \mathcal{LR}}{\partial \mathbf{w}_k} = \gamma_1 * 4 * \sum_{i \neq k} \mathbf{w}_i \mathbf{w}_i^\top \mathbf{w_k} + \gamma_2 * 2 * \mathbf{w}_k \tag{18}$$

3.5.1 Complexity analysis

During the procedure of optimization, the main cost is to calculate the loss and the gradient, and we analyse the time complexity of them respectively. For the calculation of loss, we need to first encode hash codes by linear hash functions with complexity $O(KMLN)$, then calculate empirical loss $\mathcal{LE}$ with complexity $O(KMLN_e)$, consequently calculate the regularizer with complexity $O(K^2M^2N + KML)$. For the calculation of gradient, we need to calculate the gradient of $\mathbf{H}$ with complexity $O(KMLN)$, then calculate the gradient of empirical loss with complexity $O(KM(K + L)N_e)$, finally calculate the gradient of regularizer with complexity $O(KM(KM + L)N)$. In total, the time complexity of each iteration of Gradient Descent is

$$O(KM(KM + L)N + KM(K + L)N_e) \tag{19}$$

We can see that the time complexity of each iteration is linear with the number of entities (i.e. N) and the number of links (i.e. N_e). As the $\mathbf{R}$ is usually sparse, we can optimize the final objective efficiently, and our method is scalable for large-scale training data.

As the hash functions are linear projections $\mathbf{W}$, the time complexity for generating hash code for a query is O(KML) which is very efficient.

3.6 Aggregation Strategy

For a query q, we retrieve entities from mutiple hash tables. Each returned entity has different hamming distances from query in different hash tables, which make it hard to rank these entities directly. This ranking problem mainly comes from the absence of unified ranking strategy and criteria.

One intuitive strategy is to rank the entities according to the minimum of Hamming distances from query in all hash tables. As hamming distances in each hash table only have $K+1$ discrete values (i.e. $\{0, 1, 2, \cdots, K\}$), there may be many returned entities sharing the same hamming distance from query, which make the entities not discriminative. On the other hand, the strategy also ignores some important information of multiple hash tables. For example, the more hash tables a pair of entities is close in, the more confident the similarity relationship between them is. So, we propose an aggregation strategy based on the intuitive strategy and exploits the hamming distances in all the hash tables. Particularly, we first sort the M hamming distances of each returned entity from query with ascending order; then, to determine the order of two returned entities (say i, j), we compare their sorted hamming distance list beginning from the minimum distance and forwarding until one's (say i) distance is smaller than the other's distance (say j); thus, we say that i is more similar to query q than j.

Formally, we denote the sorted hamming distance list as $hdl_{qi} = (hd_{qi}^{(1)}, hd_{qi}^{(2)}, \cdots, hd_{qi}^{(M)})$, and the order between two sorted hamming distance list (hdl_{qi}, hdl_{qj}) is defined by the following definition.

Definition 1. *Given two sorted hamming distance lists with ascending order,* $hdl_{qi} = (hd_{qi}^{(1)}, hd_{qi}^{(2)}, \cdots, hd_{qi}^{(M)})$, $hdl_{qj} = (hd_{qj}^{(1)}, hd_{qj}^{(2)}, \cdots, hd_{qj}^{(M)})$, $hdl_{qi} < hdl_{qj}$ *if and only if there exists some* l *that*

$$hd_{qi}^{(k)} = hd_{qj}^{(k)}, \quad \forall k < l$$
$$hd_{qi}^{(l)} < hd_{qj}^{(l)}$$

4. EXPERIMENTS

In this section, we will show the results of experiments on one synthetic dataset and two public datasets (i.e. DBLP and NUS-WIDE). First, we give a brief introduction of experiment setting. Then, we will report and analyse the results.

4.1 Experiment Setting

The task of the experiment is hash look up. That is, given a query, we need to search similar entities in hash table by sequentially looking up hash buckets until enough entities are found. The hash buckets are looked up in ascending order of Hamming distance between the hash codes of these buckets and the hash code of query. We select three metrics, i.e. Mean Average Precision (MAP), Precision-Recall Curve, Hamming Radius to Search (HRS), to measure the performance from different aspects. Given the number of entities, N_q, to retrieve, HRS refers to the minimum Hamming distance from query, within which we can retrieve at least N_q entities by scanning all the hash buckets. Given the number of results to search, HRS reflects the size of search space (i.e. the number of hash codes to scan), which is tightly correlated to the search efficiency. Assume the hash bit number of a hash table is K, then the number of hash codes to search is $\sum_{i=0}^{HRS} \binom{K}{i}$. So, when HRS decreases by 1, the search space will reduce by $\binom{K}{HRS}$, which is dominant in $\sum_{i=0}^{HRS} \binom{K}{i}$ when HRS is relatively small. That is, when HRS is relatively small, reduction by 1 on HRS will save most of the search space.

We select three state-of-art hashing methods, i.e. Kernel-based Supervised Hashing (KSH) [22], Semi-supervised Hashing (SPLH) [36], Iterative Quantization (ITQ) [12], as baselines. KSH is a supervised method, SPLH is a semi-supervised method, ITQ is a unsupervised methods. For KSH, we randomly select 300 anchors in NUS-WIDE, and 50 anchors in DBLP and the synthetic dataset.

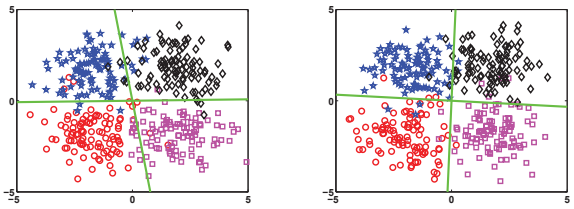
We set parameters by grid search. And we get the optimal parameters as $\gamma_1 = 1.0 \times 10^{-10}$, $\gamma_2 = 1.0 \times 10^{-7}$.

Finally, all the algorithms are implemented using Matlab. We run experiments on a machine running Windows Server 2008 with 12 2.4GHz cores and 192GB memory.

4.2 Experiments on Synthetic Data

4.2.1 Dataset

Synthetic Data is generated to simulate the situation that similarity arises from multiple components. This dataset includes 400 entities represented with 4-dimensional feature vector. The 4-dimensional feature vector consists of two 2-dimensional feature vector with each 2-dimensional feature vector corresponding to a similarity component.



(a) Similarity Component 1 (b) Similarity Component 2

Figure 3: Distribution of synthetic data on two components. There are four clusters in each similarity component. Each cluster is plotted by a specific color and marker. The green lines represent the linear hash functions of *MuCH*.

Figure 3 shows the distribution of Synthetic Data on each component respectively. In each component, there are four clusters sampled from four Gaussian distribution with centers $\{(2, 2), (2, -2), (-2, 2), (-2, -2)\}$, and each cluster contains 100 samples. In our experiments, the clusters are regarded as labels. In each component, we set the points belong to the same cluster similar, and the points belong to different clusters dissimilar. Note that the clusters in one component is independent with that in the other component. From the global view, two entities are similar when they are

similar on any similarity component, otherwise, they are dissimilar. We randomly select 300 entities as training set, and the others as testing set. Besides, we randomly select 100 entities from the training set, and set their pairwise similarities as observed.

4.2.2 Results on Synthetic Data

Figure 4 shows the hash projection matrix $\mathbf{W}$. We can see that four elements in different rows and columns are large, and the others are very small.

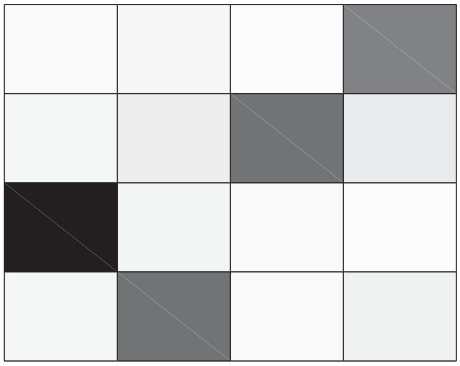


Figure 4: Hash projection matrix W. The grid in i-th row and j-th column represents W_{ij}. The darker the grid is, the magnitude of W_{ij} is larger. As the last two rows of $\mathbf{W}^{(1)}$ (first two columns) is large and the first two row is small, Similarity Component 2 can be extracted by $\mathbf{W}^{(1)}$. In a similar way, $\mathbf{W}^{(2)}$ (last two columns) can extract Similarity Component 1.

So, the two large elements in the first two rows of $\mathbf{W}^{(2)}$ can extract the first two dimensions of the feature vector, i.e. Similarity Component 1; and the two large elements in the last two rows of $\mathbf{W}^1$ extract the last two dimensions of the feature vector, i.e. Similarity Component 2. That is, *MuCH* can accurately identify the similarity components in the synthetic data. Figure 3 shows the linear hash projections, $\mathbf{W}^{(2)}$ and $\mathbf{W}^{(1)}$, in two similarity components, respectively. We can see that the clusters are separated well by hash projections.

Table 1: Performance on Synthetic Data, where F1 is measured on the results that share the common hash buckets with queries. All the methods use 4-bit hash codes. *MuCH* learns two 2-bit hash tables.

Method	MuCH	KSH	SPLH	ITQ
MAP	**0.9380**	0.8275	0.6993	0.8581
F1	**0.9310**	0.2772	0.2359	0.2514

Table 1 shows the quantative performance on Synthetic Data of all methods. Although the other methods can also learn hash functions adaptive to the data distribution, but they cannot identify the underlying similarity components. Thus, these methods may split entity pairs that are similar on just one component, and can only retrieve the entities that are fully similar, i.e. similar on both similarity components. So, the F1 score is very small. As the comparative methods split the partially similarity pairs and mix the similar and dissimilar entities up, the MAPs of them are much smaller than that of *MuCH*.

4.3 Experiments on Public Datasets

4.3.1 Datasets

NUS-WIDE [8] is an image dataset crawled from Flickr with about 260,000 images and 81 concept categories. A pair of images is regarded similar if they share at least one common concept, otherwise, they are dissimilar. As many images are labeled by multiple concepts, the similarity between images is non-transitive. We use the top 10 concepts in our experiments, and get about 180,000 images. We randomly select 10000 images as training set, and 5000 images as testing set. Besides, we randomly sample 1000 images from training set, and set the pairwise similarities between them as observed. The 500-dimeansional Bag-of-Visual-Words (BOW) of SIFT are used as feature.

DBLP[2] is a digital bibliography of computer science community. We select the authors that have published at least 3 papers in 12 specific fields which include 33 corresponding conferences (see Table 2). After the selection, we get about 3500 authors, and about 20000 papers. A pair of authors is regarded similar if they have published papers in at least one common field, otherwise, they are dissimilar. As an author may publish papers in multiple fields, the similarity between authors is non-transitive. We aggregate the paper contents (title and abstract) of each user as a document, then Latent Dirichlet Allocation (LDA) [3] is performed on the documents set to get the distribution of authors on 100 topics. We use the 100-dimensional distribution on LDA topics as feature. 2500 authors are selected as training set, and the others are used as testing set. Besides, we select 200 authors from the training set, and set the pairwise similarities between them observed.

Table 2: Selected research fields and corresponding conferences in DBLP

Field	Conference
Database	ICDE, VLDB, SIGMOD, PODS, EDBT
Data Mining	KDD, ICDM, SDM, PKDD, PAKDD
Artificial Intelligence	IJCAI, AAAI
Information Retrieval	SIGIR, ECIR
Computer Vision	CVPR
Machine Learning	ICML, ECML
Algorithms & Theory	STOC, FOCS, SODA, COLT
Natural Language Processing	ACL, ANLP, COLING
Bioinformatics	ISMB, RECOMB
Networking	SIGCOMM, MOBICOM, INFOCOM
Operating Systems	SOSP, OSDI
Distributed & Parallel Computing	PODC, ICS

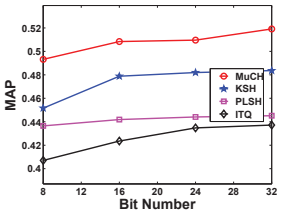
[2] `http://www.informatik.uni-trier.de/~ley/db/`

Table 3: Retrieval performance evaluated by MAP and corresponding Hamming Radius to Search (HRS). We retrieve 500 nearest entities for evaluation. For MuCH-S, the bit number in second row represents the bit number of single hash table, and the number of hash tables is 4. For MuCH-F, the bit number in second row represents the total bit number of all hash tables, and the number of hash tables is 2.

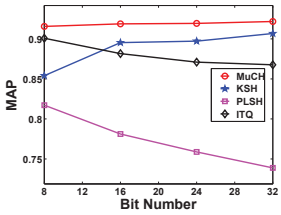
Method	NUS-WIDE				DBLP			
	16 bits		32 bits		16 bits		32 bits	
	MAP	HRS	MAP	HRS	MAP	HRS	MAP	HRS
MuCH-S	**0.5084**	2.0658	**0.5191**	5.6182	**0.9188**	3.172	**0.9217**	8.108
MuCH-F	0.4863	**0.7783**	0.5051	**2.4708**	0.9108	**1.8640**	0.9151	**3.7340**
KSH	0.4788	2.8595	0.4836	6.1010	0.8808	4.5020	0.8866	9.3560
SSH	0.4419	1.6221	0.4451	3.7725	0.7812	6.2000	0.7389	13.6720
ITQ	0.4236	3.4185	0.4372	8.5899	0.8816	6.4260	0.8677	13.8980

4.3.2 *Results on NUS-WIDE and DBLP*

Table 3 shows the retrieval performance on both public datasets. We can see that the MAPs of MuCH-S achieve at least 3% absolute improvement over the baselines. Although MuCH-S cost four times more storage than the others, but its search efficiency (HRS) is still comparative to (even a little higher than) the baselines (MuCH-F is our method). MuCH-F cost the same storage as the baselines and achieve similar search performance (MAP) as the baselines. But, MuCH-F achieves much higher efficiency. On NUS-WIDE, HRS (i.e. Hamming Radius to Search) of *MuCH* is at least 0.844 smaller in 16 bits setting and 1.3 smaller in 32 bits setting than the others. On DBLP, HRS of *MuCH* is at least 1.38 smaller on 16 bits setting and 3.43 smaller on 32 bits setting than the others. According to the above comparisons with baselines, we can see that *MuCH* can achieve much higher search accuracy with comparative search efficiency and perform much more efficiently with same storage. This means *MuCH* learns more effective representation for data with non-transitive similarity than the competitive hashing methods.



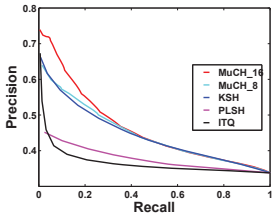
(a) NUS-WIDE



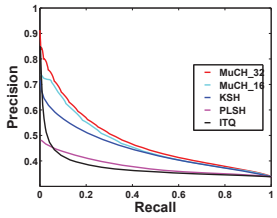
(b) DBLP

Figure 5: MAP on different hash bit numbers, $\{8, 16, 24, 32\}$. The left figure is the results on NUS-WIDE, and the right is the results on DBLP. For *MuCH*, the hash bit number represents the bit number of each hash table, and the number of hash tables is 4.

Figure 5 shows the MAPs on different hash bit numbers on NUS-WIDE and DBLP. Figure 6 and Figure 7 shows the Precision-Recall curves on NUS-WIDE and DBLP. Different from the above comparison, in Figure 5, the hash bit number of single hash table in *MuCH* is equal to that for the comparative methods. Although this will increase three times more storage, the search efficiency (i.e. HRS) is still comparable to other methods (comparing the HRS of *MuCH* on 32 bits with the HRS of other methods on 16 bits in Table 3), and the retrieval accuracy (MAP) of *MuCH* outperforms the best of other methods by 7.1% on NUS-WIDE and 3.4% on DBLP. We can see that *MuCH* is also better than comparative methods on Precision-Recall curves in Figure 6 and Figure 7.

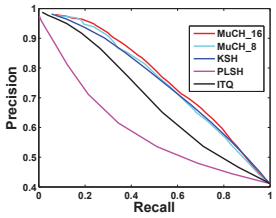


(a) 16 bits

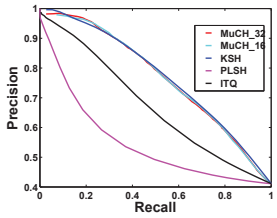


(b) 32 bits

Figure 6: Precision-Recall curve on NUS-WIDE. From left to right, the methods in two figures use 16 hash bits and 32 hash bits respectively. MuCH_k means *MuCH* learning two k-bit hash tables.



(a) 16 bits



(b) 32 bits

Figure 7: Precision-Recall curve on DBLP. From left to right, the methods in two figures use 16 hash bits and 32 hash bits respectively. MuCH_k means *MuCH* learning two k-bit hash tables.

4.3.3 *Insights on MuCH*

Figure 8 and Figure 9 shows the retrieval performance of *MuCH* in different settings by varying hash bit number and hash table number. Fixing the hash bit number, in most situations, MAP increases monotonously with respect to hash table number. This means that we can improve retrieval accuracy by adding more hash tables. Compared with increasing hash bits in single hash table, this approach can preserve the efficiency because the search time just increases linearly with respect to the hash table number. On the other

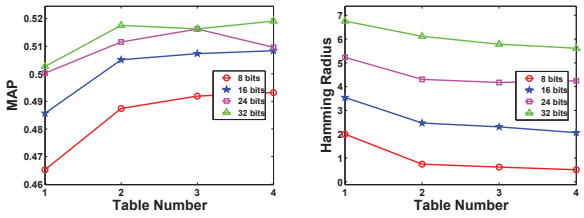


Figure 8: Retrieval performance of *MuCH* on NUS-WIDE. Different hash bit number of each hash table, $\{8, 16, 24, 32\}$ and different hash tables number, $\{1, 2, 3, 4\}$ are tested. MAP and corresponding HRS are used as evaluation metrics.

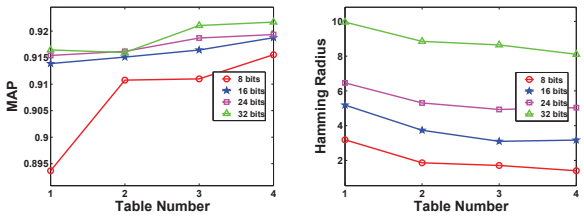


Figure 9: Retrieval performance of *MuCH* on DBLP. Different hash bit number of each hash table, $\{8, 16, 24, 32\}$ and different hash table number, $\{1, 2, 3, 4\}$ are tested. MAP and corresponding HRS are used as evaluation metrics.

hand, HRS decreases monotonously with respect to the hash table number. And the decrease of HRS achieves about 2 in some situations, which will improve the search efficiency significantly. The reason may be that the number of similarity components, which each hash table corresponds to, decreases with the hash table number increasing. Then, the non-transitive similarities can be approximated more accurately, and the similar entities will be aggregated more close.

When the hash table number is fixed, MAP increases monotonously with respect to hash bit number, and HRS also increases monotonously with respect to hash bit number. Therefore, when hash table number is fixed, we need to make a tradeoff between the retrieval accuracy and efficiency.

5. CONCLUSION

In this paper, we argue that non-transitive similarity due to various latent similarity components is a ubiquitous phenomenon in many real-world applications, including image retrieval, document search, recommendation system and so on. Many existing hashing learning methods employ the pairwise similarity as supervised information, while neglecting the non-transitivity of those similarity relationships. In this paper, we propose a novel hashing method, called Multi-Component Hashing (*MuCH*), to capture the latent similarity components to handle the non-transitive property. *MuCH* employs linear hash functions to project data into multiple hash tables, with each hash table corresponding to a latent similarity component, by which the non-transitive similarity can be maintained across different hash tables. Given a query, *MuCH* generates multiple hash codes to retrieves similar entities from each hash table of the database points. Then the returned results are organized though using a specific aggregation strategy to generate the final search results. Extensive experiments on both synthetic and real benchmark datasets shows that our method outperforms several representative hashing techniques on both accuracy and efficiency

One of our future directions is to leverage multiple-view and multiple-modality data sources to further improve the performance though identifying more discriminant similarity components.

6. ACKNOWLEDGMENTS

This work is supported by the National Basic Research Program of China, No. 2015CB352300; National Program on Key Basic Research Project, No. 2015CB352300; National Natural Science Foundation of China, No. 61370022 and No. 61210008; International Science and Technology Cooperation Program of China, No. 2013DFG12870. Thanks for the support of NExT Research Center funded by MDA, Singapore, under the research grant, WBS:R-252-300-001-490 and the research fund of Tsinghua-Tencent Joint Laboratory for Internet Innovation Technology.

7. REFERENCES

[1] I. Abraham, S. Chechik, D. Kempe, and A. Slivkins. Low-distortion inference of latent similarities from a multiplex social network. In *SODA*, pages 1853–1872. SIAM, 2013.

[2] E. M. Airoldi, D. M. Blei, S. E. Fienberg, and E. P. Xing. Mixed membership stochastic blockmodels. *Journal of Machine Learning Research*, 9(1981-2014):3, 2008.

[3] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003.

[4] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise independent permutations. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 327–336, 1998.

[5] M. Bronstein, A. Bronstein, F. Michel, and N. Paragios. Data fusion through cross-modality metric learning using similarity-sensitive hashing. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3594–3601, 2010.

[6] S. Chang, G. Qi, C. C. Aggarwal, J. Zhou, M. Wang, and T. S. Huang. Factorized similarity learning in networks. In *2014 IEEE International Conference on Data Mining, ICDM 2014, Shenzhen, China, December 14-17, 2014*, pages 60–69, 2014.

[7] S. Changpinyo, K. Liu, and F. Sha. Similarity component analysis. In *Advances in Neural Information Processing Systems*, pages 1511–1519, 2013.

[8] T.-S. Chua, J. Tang, R. Hong, H. Li, Z. Luo, and Y. Zheng. Nus-wide: a real-world web image database from national university of singapore. In *Proceedings of the ACM International Conference on Image and Video Retrieval*, page 48, 2009.

[9] P. Cui, S.-W. Liu, W.-W. Zhu, H.-B. Luan, T.-S. Chua, and S.-Q. Yang. Social-sensed image search. *ACM Transactions on Information Systems (TOIS)*, 32(2):8, 2014.

[10] S. E. Fienberg, M. M. Meyer, and S. S. Wasserman. Statistical analysis of multiple sociometric relations. *Journal of the american Statistical association*, 80(389):51–67, 1985.

[11] Y. Gong, S. Kumar, V. Verma, and S. Lazebnik. Angular quantization-based binary codes for fast similarity search. In *Advances in Neural Information Processing Systems*, pages 1205–1213, 2012.

[12] Y. Gong and S. Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In *IEEE*

Conference on Computer Vision and Pattern Recognition, pages 817–824, 2011.

[13] J. Hopcroft, T. Lou, and J. Tang. Who will follow you back?: reciprocal relationship prediction. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 1137–1146. ACM, 2011.

[14] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613, 1998.

[15] M. Jiang, P. Cui, F. Wang, Q. Yang, W. Zhu, and S. Yang. Social recommendation across multiple relational domains. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 1422–1431. ACM, 2012.

[16] M. Jiang, P. Cui, F. Wang, W. Zhu, and S. Yang. Scalable recommendation with social contextual information. *Knowledge and Data Engineering, IEEE Transactions on*, 26(11):2789–2802, 2014.

[17] W. Kong and W.-J. Li. Isotropic hashing. In *Advances in Neural Information Processing Systems*, pages 1655–1663, 2012.

[18] S. Kumar and R. Udupa. Learning hash functions for cross-view similarity search. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, pages 1360–1365, 2011.

[19] G. Lin, C. Shen, Q. Shi, A. van den Hengel, and D. Suter. Fast supervised hashing with decision trees for high-dimensional data. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 1971–1978. IEEE, 2014.

[20] G. Lin, C. Shen, D. Suter, and A. van den Hengel. A general two-step approach to learning-based hashing. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 2552–2559. IEEE, 2013.

[21] S. Liu, P. Cui, H. Luan, W. Zhu, S. Yang, and Q. Tian. Social-oriented visual image search. *Computer Vision and Image Understanding*, 118:30–39, 2014.

[22] W. Liu, J. Wang, R. Ji, Y.-G. Jiang, and S.-F. Chang. Supervised hashing with kernels. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2074–2081, 2012.

[23] W. Liu, J. Wang, S. Kumar, and S.-F. Chang. Hashing with graphs. In *Proceedings of the 28th International Conference on Machine Learning*, pages 1–8, 2011.

[24] X. Liu, J. He, and B. Lang. Reciprocal hash tables for nearest neighbor search. In *Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.

[25] X. Liu, Y. Mu, B. Lang, and S.-F. Chang. Compact hashing for mixed image-keyword query over multi-label images. In *Proceedings of the 2nd ACM International Conference on Multimedia Retrieval*, page 18. ACM, 2012.

[26] Y. Mu, X. Chen, T.-S. Chua, and S. Yan. Learning reconfigurable hashing for diverse semantics. In *Proceedings of the 1st ACM International Conference on Multimedia Retrieval*, page 7. ACM, 2011.

[27] Y. Mu and S. Yan. Non-metric locality-sensitive hashing. In *AAAI*, 2010.

[28] M. Ou, P. Cui, F. Wang, J. Wang, W. Zhu, and S. Yang. Comparing apples to oranges: a scalable solution with heterogeneous hashing. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 230–238. ACM, 2013.

[29] M. Ou, P. Cui, J. Wang, F. Wang, and W. Zhu. Probabilistic attributed hashing. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[30] R. Salakhutdinov and G. Hinton. Semantic hashing. *RBM*, 500(3):500, 2007.

[31] J. Song, Y. Yang, Y. Yang, Z. Huang, and H. T. Shen. Inter-media hashing for large-scale retrieval from heterogeneous data sources. In *Proceedings of the 2013 international conference on Management of data*, pages 785–796. ACM, 2013.

[32] M. Szell, R. Lambiotte, and S. Thurner. Multirelational organization of large-scale social networks in an online world. *Proceedings of the National Academy of Sciences*, 107(31):13636–13641, 2010.

[33] L. van der Maaten and G. Hinton. Visualizing non-metric similarities in multiple maps. *Machine learning*, 87(1):33–55, 2012.

[34] J. Wang, S. Kumar, and S.-F. Chang. Sequential projection learning for hashing with compact codes. In *Proceedings of International Conference on Machine Learning*, pages 1127–1134, 2010.

[35] J. Wang, S. Kumar, and S.-F. Chang. Semi-supervised hashing for large-scale search. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(12):2393–2406, 2012.

[36] J. Wang, S. Kumar, and S.-F. Chang. Semi-supervised hashing for large-scale search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(12):2393–2406, 12 2012.

[37] Z. Wang, W. Zhu, P. Cui, L. Sun, and S. Yang. Social media recommendation. In *Social Media Retrieval*, pages 23–42. Springer, 2013.

[38] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *Advances in Neural Information Processing Systems*, pages 1753–1760, 2008.

[39] H. Xu, J. Wang, Z. Li, G. Zeng, S. Li, and N. Yu. Complementary hashing for approximate nearest neighbor search. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 1631–1638. IEEE, 2011.

[40] D. Zhang, F. Wang, and L. Si. Composite hashing with multiple information sources. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 225–234. ACM, 2011.

[41] Y. Zhen and D. Yeung. A probabilistic model for multimodal hash function learning. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 940–948, 2012.