OPTIMIZING AFFINITY-BASED BINARY HASHING USING AUXILIARY COORDINATES

Ramin Raziperchikolaei and **Miguel Á. Carreira-Perpiñán**, EECS, UC Merced

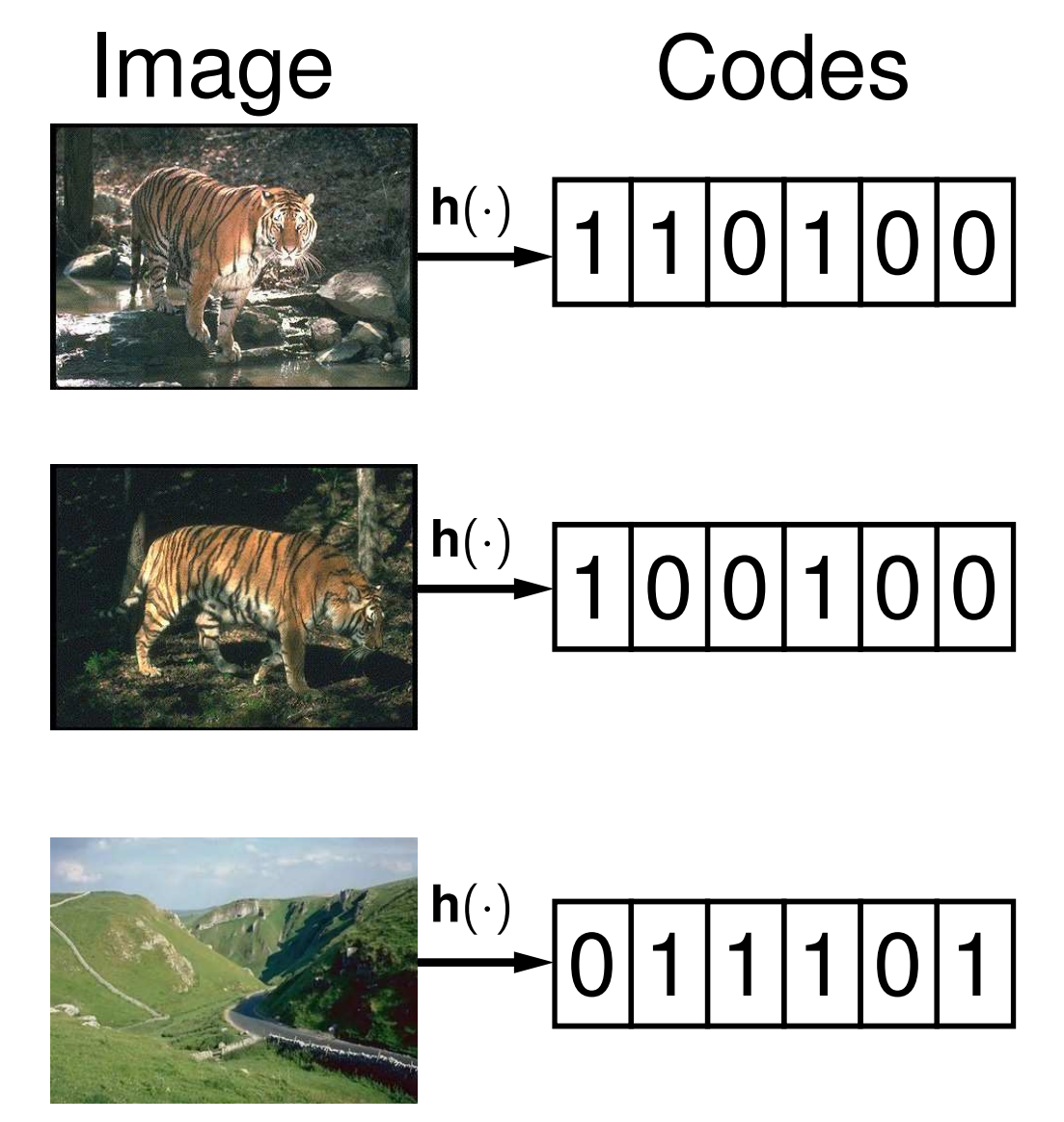
1 Abstract

Binary hashing is a well-known approach for fast approximate nearest-neighbor search in information retrieval. Much work has focused on affinity-based objective functions. This typically results in a difficult nonconvex and nonsmooth optimization problem. Much work has simply relaxed the problem, solving a continuous optimization, and truncating the codes, or first optimized the objective over the binary codes and then learned the hash function a posteriori. Both approaches are suboptimal. We propose a general framework that optimizes affinity-based objective jointly over the hash functions and the binary codes so that they gradually match each other. Our optimization is guaranteed to obtain better hash functions while being not much slower. In addition, our framework facilitates the design of optimization algorithms for arbitrary types of loss and hash functions. Work supported by NSF award IIS–1423515

2 Binary hash functions for fast image retrieval

In K nearest neighbors problem, there are N training points in D-dimensional space (usually $D > 100$) $\mathbf{x}_i \in \mathbb{R}^D, i = 1, \dots, N$. The goal is to find the K nearest neighbors of a query point $\mathbf{x}_q \in \mathbb{R}^D$. Exact search in the original space is $\mathcal{O}(ND)$ in time and space.

A binary hash function $\mathbf{h}$ takes as input a high-dimensional vector $\mathbf{x} \in \mathbb{R}^D$ and maps it to an b-bit vector $\mathbf{z} = \mathbf{h}(\mathbf{x}) \in \{0, 1\}^b$.
The main goal is preserving the neighborhood, i.e., assign (dis)similar codes to (dis)similar patterns.
In supervised hashing, we try to preserve tshe semantic similarity between the images (e.g. images from different view points are similar, while they are far in the Euclidean space).



Finding K nearest neighbors in Hamming space needs $\mathcal{O}(Nb)$ in time and space. Distances can be computed efficiently using hardware operations.

$N = 10^9$, $D = 500$ and $b = 64$

Search in	Space	Time
Original space	2 TB	1 hour
Hamming space	8 GB	10 seconds

3 Affinity-based objective functions

Most hashing papers try to minimize an affinity-based objective function, which directly tries to preserve the original similarities in the binary space:

$$\min \mathcal{L}(\mathbf{h}) = \textstyle\sum_{n,m=1}^{N} L(\mathbf{h}(\mathbf{x}_n), \mathbf{h}(\mathbf{x}_m);\ y_{nm}) \qquad (1)$$

where $\mathbf{x}_i \in \mathbb{R}^D$ is the ith input data, $\mathbf{h}$ is the parameters of the hash function, $L(\cdot)$ is a loss function that compares the codes for two images with the ground-truth value y_{nm} that measures the affinity in the original space between the two images $\mathbf{x}_n$ and $\mathbf{x}_m$. Many such loss functions $L(\mathbf{z}_n, \mathbf{z}_m;\ y_{nm})$ exist, e.g.:
KSH: $(\mathbf{z}_n^T\mathbf{z}_m - by_{nm})^2$ Laplacian: $(y_{nm} \|\mathbf{z}_n - \mathbf{z}_m\|^2)$ BRE: $(\frac{1}{b}\|\mathbf{z}_n - \mathbf{z}_m\|^2 - y_{nm})^2$

4 Optimization-based approaches

Optimizing $\mathcal{L}(\mathbf{h})$ is difficult because $\mathbf{h}$ is discrete. Many optimization algorithms have been considered in the binary hashing literature:

- Relaxation (Liu et al., 2012): relax the step function or binary codes (ignore the binary nature of the problem), optimize the objective continuously, and truncate the results.
- Two-step methods (Lin et.al., 2013, 2014): in the first step, define the objective over binary codes, optimize it approximately and learn the codes, and in the second step, fit the hash function given the codes.

Limitations of optimization-based methods:

- The hash function outputs binary values, hence the problem is nonconvex and nonsmooth. The underlying problem of finding the binary codes for the points is an NP-complete optimization over Nb variables.
- Most methods do not scale beyond a few thousand training points.
- The b single-bit hash functions are coupled (to avoid trivial solutions where all codes are the same).
- In the end, there is little practical difference between the different objective functions and optimization algorithms proposed.

5 Optimization using the method of auxiliary coordinates

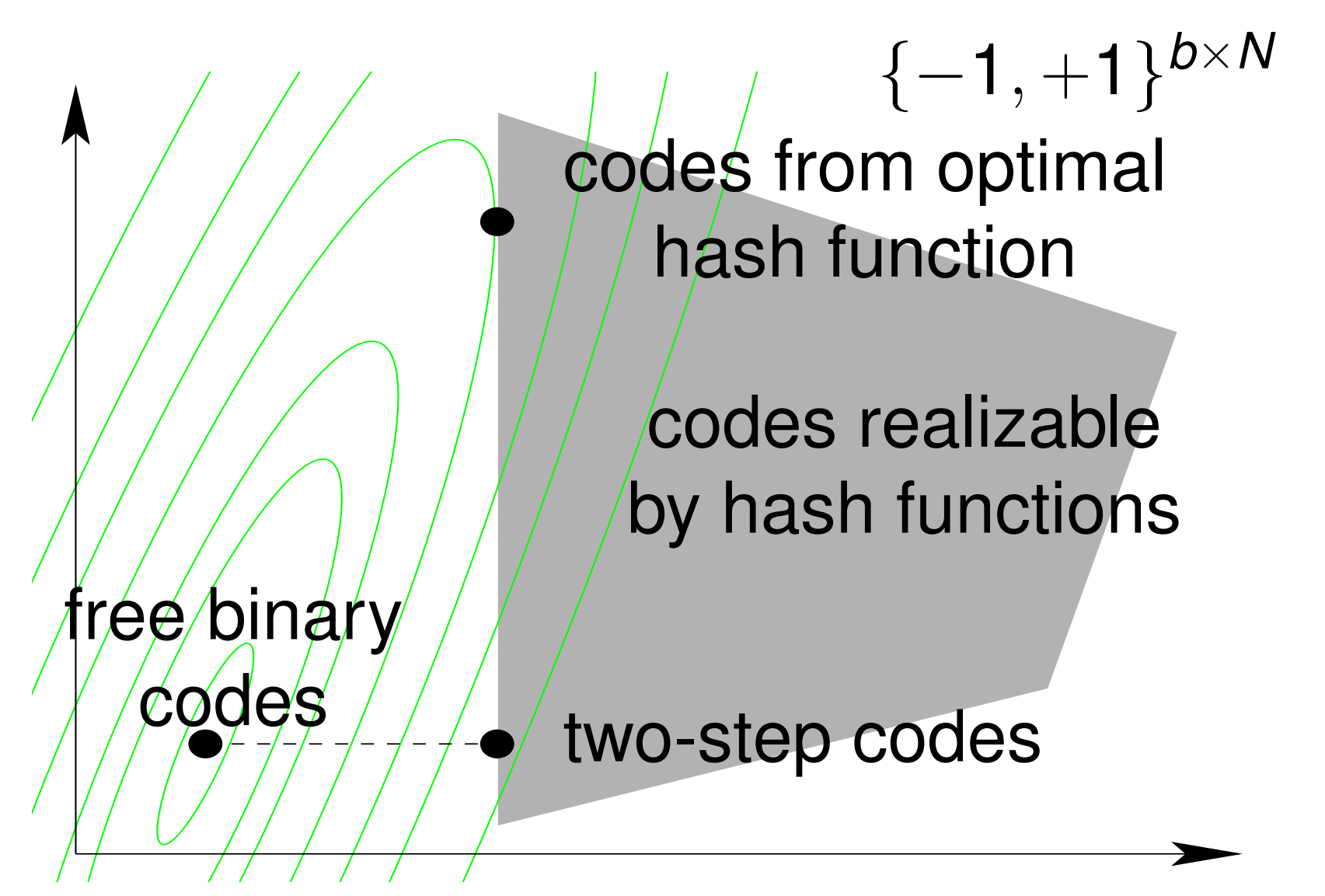
We show how to optimize the objective in eq. (1) correctly by preserving the binary constraints and considering all elements of the problem.

We use the method of auxiliary coordinates (MAC) (Carreira-Perpiñán and Wang, 2004), a generic approach to optimize nested functions. Our proposed method has three main steps:

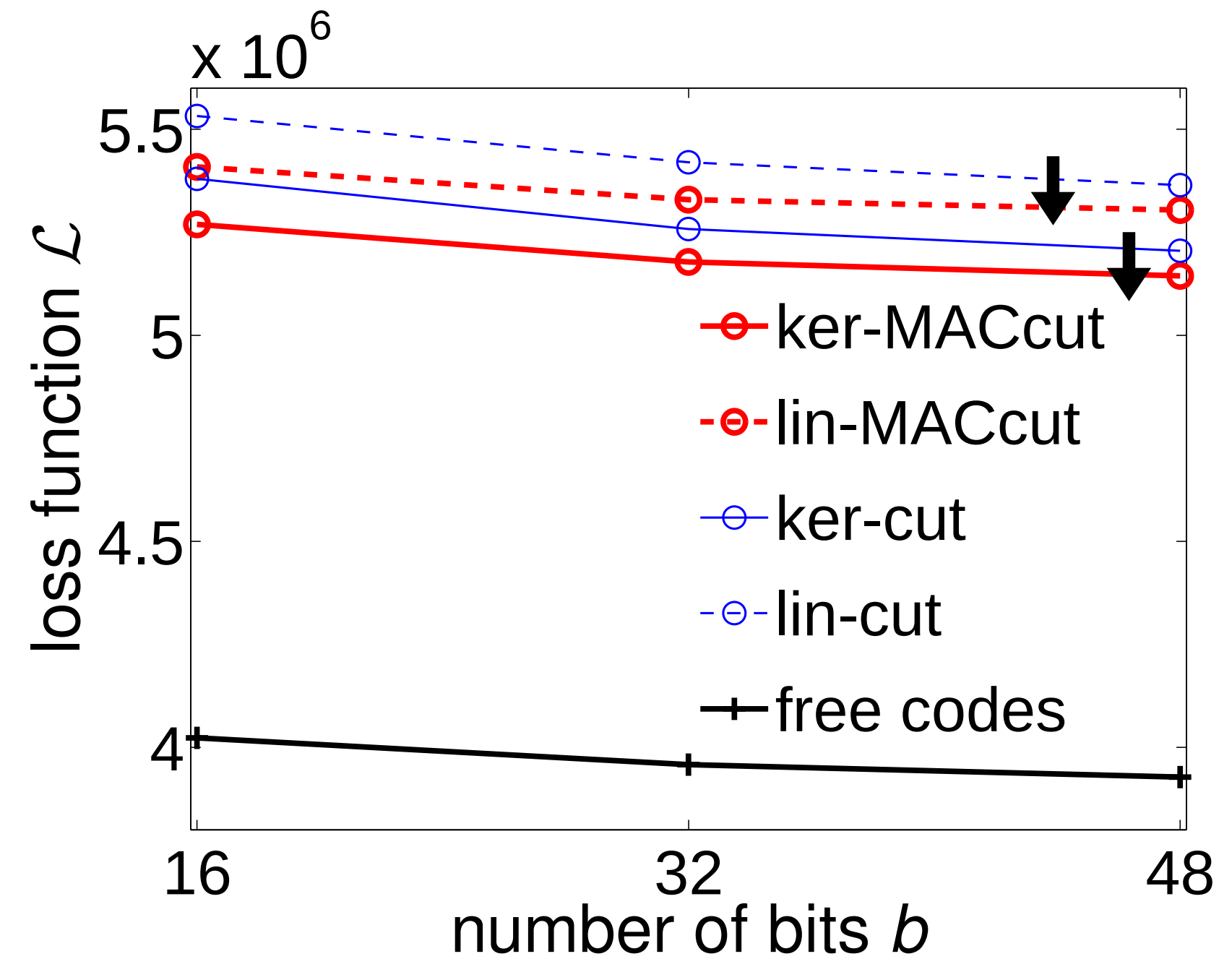
1. we introduce auxiliary coordinates $\mathbf{z}_n \in \{-1, +1\}^b$ as the output of $\mathbf{h}(\mathbf{x}_n)$ and convert the problem for $\mathcal{L}(\mathbf{h})$ into an equivalent constrained problem:
$$\mathcal{L}_C(\mathbf{h}, \mathbf{Z}) = \textstyle\sum_{n,m=1}^{N} L(\mathbf{z}_n, \mathbf{z}_m;\ y_{nm}) \text{ s.t. } \mathbf{z}_1 = \mathbf{h}(\mathbf{x}_1), \cdots, \mathbf{z}_N = \mathbf{h}(\mathbf{x}_N)$$
2. Now we apply the quadratic-penalty method:
$$\mathcal{L}_P(\mathbf{h}, \mathbf{Z}; \mu) = \textstyle\sum_{n,m=1}^{N} L(\mathbf{z}_n, \mathbf{z}_m;\ y_{nm}) + \mu \sum_{n=1}^{N} \|\mathbf{z}_n - \mathbf{h}(\mathbf{x}_n)\|^2$$
where $\mathbf{z}_1, \dots, \mathbf{z}_N \in \{-1, +1\}^b$. We start with a small μ and increase it slowly.
3. To optimize $\mathcal{L}_P(\mathbf{h}, \mathbf{Z}; \mu)$, we apply alternating optimization:
   - Optimization over $\mathbf{h}$ given $\mathbf{Z}$: $\min_{\mathbf{h}} \sum_{n=1}^{N} \|\mathbf{z}_n - \mathbf{h}(\mathbf{x}_n)\|^2$. This is equivalent to training b binary classifiers with data $(\mathbf{X}, \mathbf{Z})$.
   - Optimization over $\mathbf{Z}$ given $\mathbf{h}$: this is an NP-complete problem over Nb binary variables. We apply alternating optimization over the ith bit of points given the rest are fixed. This gives a binary quadratic problem.

If we optimize the objective $\mathcal{L}_P$ for $\mu \to 0^+$, we get an algorithm that first optimizes the codes and then fits the hash function a posteriori. MAC starts from the results of this algorithm (two-step methods).

6 Experiments: why does MAC learn better codes and hash functions?

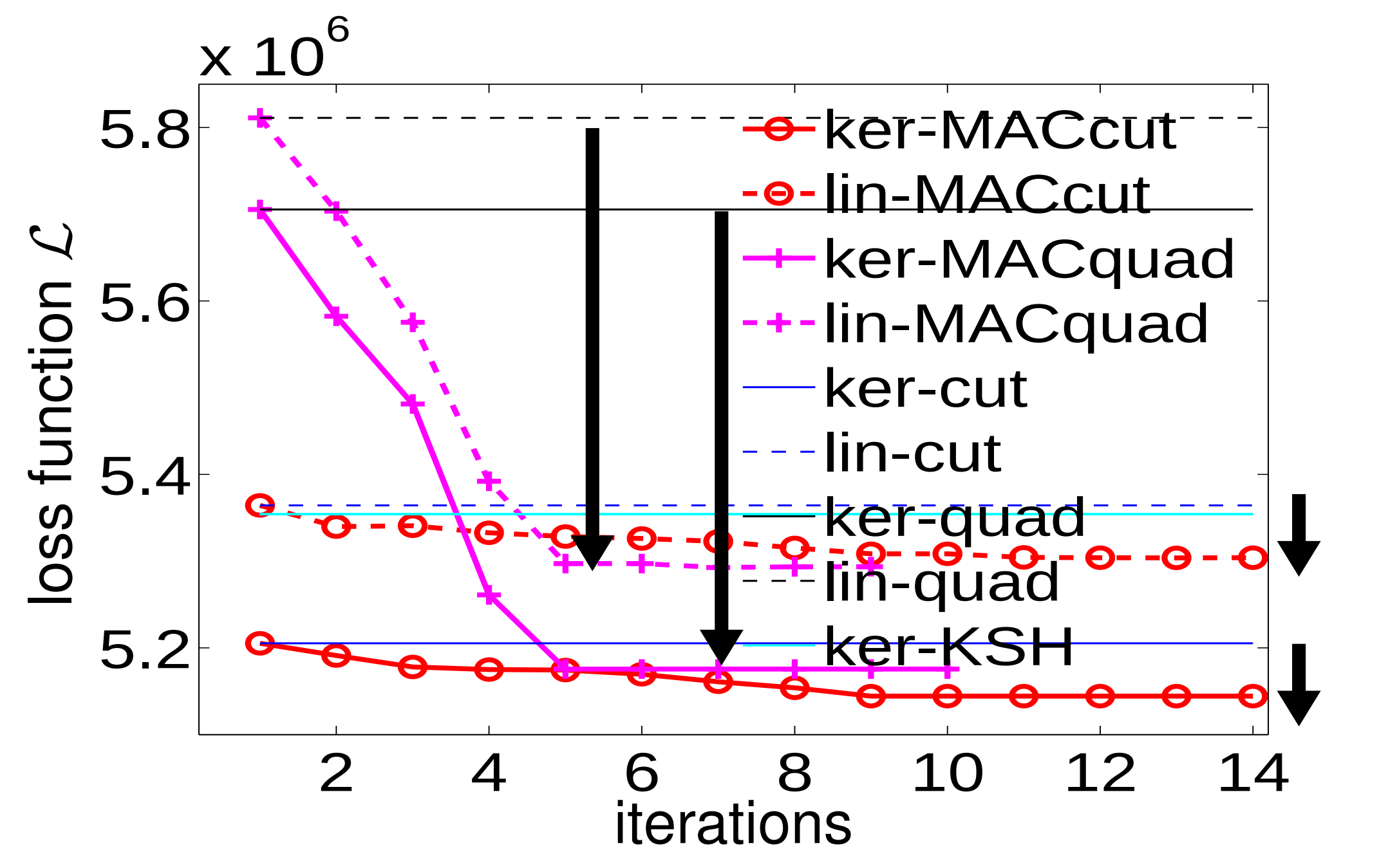

$\{-1,+1\}^{b\times N}$
codes from optimal hash function
codes realizable by hash functions
free binary codes
two-step codes

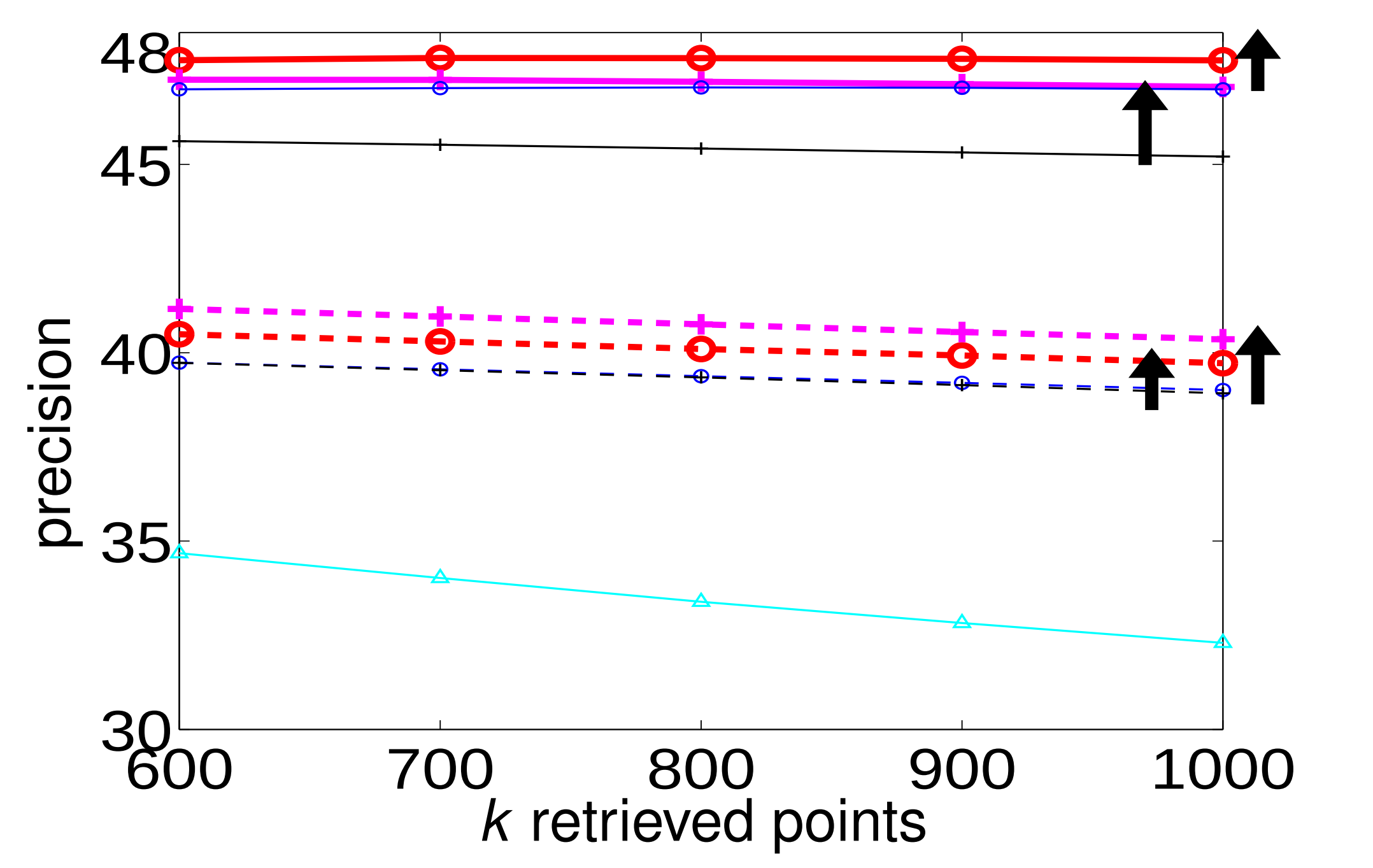
This figure shows the space of all possible binary codes and the feasible set for linear hash functions. The contours correspond to $\mathcal{L}_c$ defined only on codes.
The two-step method projects the free codes into the feasible set.
MAC optimizes the codes and functions jointly to find a better local optima.



We compare the codes of cut with MACcut, in CIFAR dataset. We achieve free codes by minimizing $\mathcal{L}_c$ over the binary codes $\mathbf{Z}$ without any constraint. Free codes are the starting point of both cut and MACcut. Free codes always achieve lower error than the cut and MACcut.

MAC achieves lower error than the cut using both linear and kernel hash function and using different loss functions.





"cut" (Lin et al., 2014) and "quad" (Lin et al., 2013) are two-step methods that first learn the binary codes and then train the hash function on them. They optimize the binary codes using a quadratic surrogate and a GraphCut method, respectively.
MAC uses cut (MACcut) or quad (MACquad) to learn the codes in the Z-step.

MAC achieves lower error and better retrieval results using different types of hash functions and loss functions, in different datasets.

We compare MAC with cut and quad on CIFAR dataset using $b = 48$ bits. MAC finds hash functions with significantly lower objective function values than the previous approaches. It also achieves better precision.

7 Theoretical results

We can prove the following under the assumption that the $\mathbf{Z}$ and $\mathbf{h}$ steps are exact:

1. The MAC algorithm stops after a finite number of iterations, when $\mathbf{Z} = \mathbf{h}(\mathbf{X})$ in the $\mathbf{Z}$ step, since then the constraints are satisfied and no more changes will occur to $\mathbf{Z}$ or $\mathbf{h}$.
2. The path over the continuous penalty parameter $\mu \in [0, \infty)$ is in fact discrete: the minimizer $(\mathbf{h}, \mathbf{Z})$ of $\mathcal{L}_P$ for $\mu \in [0, \mu_1]$ is identical to the minimizer for $\mu = 0$, and the minimizer for $\mu \in [\mu_2, \infty)$ is identical to the minimizer for $\mu \to \infty$, where $0 < \mu_1 < \mu_2 < \infty$. Besides, the interval $[\mu_1, \mu_2]$ is itself partitioned in a finite set of intervals and the minimizer changes only at interval boundaries.

8 Conclusion

- MAC optimizes the desired objective in eq. (1) by respecting the binary nature of the codes. It transforms the original problem into a problem over codes and the hash function, and optimizes it jointly over all the parameters and finds better local optima.
- MAC performs better than the previous methods in both optimization and information retrieval measures.
- Our framework makes it easy to design an optimization algorithm for a new choice of loss function or hash function.
- See also Independent Laplacian Hashing (NIPS 2016) which uses ensemble diversity approach to learn the single-bit hash functions independently. This makes the optimization faster and simpler.