Supplementary Material for "Compact Hyperplane Hashing with Bilinear Functions"

Wei Liu[†] **Jun Wang**[‡] **Yadong Mu**[†] **Sanjiv Kumar**[§] **Shih-Fu Chang**[†]

[†]Columbia University, New York, NY 10027, USA {wliu,muyadong,sfchang}@ee.columbia.edu

[‡]IBM T. J. Watson Research Center, Yorktown Heights, NY 10598, USA wangjun@us.ibm.com

[§]Google Research, New York, NY 10011, USA sanjivk@google.com

1. Theoretical Guarantees

Theorem 2 in the main paper introduces the asymptotic complexity of our proposed randomized bilinear hashing scheme BH-Hash. For self-contained consideration, here we provide the proof, which is basically following the technique previously used in the proof of Theorem 1 in (Gionis et al., 1999). Recall that in Theorem 1 of the main paper, we have shown that the proposed BH-Hash is $(r, r(1+\epsilon), p_1, p_2)$-sensitive. Particularly, we show that $p_1 = \frac{1}{2} - \frac{2r}{\pi^2}$ and $p_2 = \frac{1}{2} - \frac{2r(1+\epsilon)}{\pi^2}$ for BH-Hash.

Theorem 2. Suppose we have a database $\mathcal{X}$ of n points. Denote the parameters $k = \log_{1/p_2} n$, $\rho = \frac{\ln p_1}{\ln p_2}$, and $c \geq 2$. Given a hyperplane query $\mathcal{P}_{\boldsymbol{w}}$, if there exists a database point $\boldsymbol{x}^*$ such that $\mathcal{D}(\boldsymbol{x}^*, \mathcal{P}_{\boldsymbol{w}}) \leq r$, then the BH-Hash algorithm is able to return a database point $\hat{\boldsymbol{x}}$ such that $\mathcal{D}(\hat{\boldsymbol{x}}, \mathcal{P}_{\boldsymbol{w}}) \leq r(1+\epsilon)$ with probability at least $1 - \frac{1}{c} - \frac{1}{e}$ by using n^ρ hash tables of k hash bits each. The query time is dominated by $O(n^\rho \log_{1/p_2} n)$ evaluations of the hash functions from $\mathcal{B}$ and cn^ρ computations of the pairwise distances $\mathcal{D}$ between $\mathcal{P}_w$ and the points hashed into the same buckets.

Proof. Denote the number of hash tables to be L. For the l-th hash table, the proposed BH-Hash algorithm randomly samples k hash functions $h^{\mathcal{B}}_{l,1}, \cdots, h^{\mathcal{B}}_{l,k}$ with replacement from $\mathcal{B}$, which will generate a k-bit hash key for each input data vector $\boldsymbol{x}$. We denote $\boldsymbol{x}$'s hash code by $H^{\mathcal{B}}_l(\boldsymbol{x}) = [h^{\mathcal{B}}_{l,1}(\boldsymbol{x}), \cdots, h^{\mathcal{B}}_{l,k}(\boldsymbol{x})]$. The main observation is that using $L = n^\rho$ independent hash tables, a $(1+\epsilon)$-approximate nearest neighbor is achieved with a non-trivial constant probability. Moreover, the query (search) time complexity is proved to be sub-linear with respect to the entire data number n.

To complete the proof, we define the following two events $\mathbf{F}_1$ and $\mathbf{F}_2$. It suffices to prove the theorem by showing that both $\mathbf{F}_1$ and $\mathbf{F}_2$ hold with probability larger than 0.5. The two events are defined as below:

$\mathbf{F}_1$: If there exists a database point $\boldsymbol{x}^*$ such that $\mathcal{D}(\boldsymbol{x}^*, \mathcal{P}_{\boldsymbol{w}}) \leq r$, then $H^{\mathcal{B}}_l(\boldsymbol{x}^*) = H^{\mathcal{B}}_l(\mathcal{P}_{\boldsymbol{w}})$ for some $1 \leq l \leq L$.

$\mathbf{F}_2$: Provided with a false alarm set

$$\mathcal{S} = \left\{ \check{\boldsymbol{x}} \;\middle|\; \check{\boldsymbol{x}} \in \mathcal{X} \text{ such that } \mathcal{D}(\check{\boldsymbol{x}}, \mathcal{P}_{\boldsymbol{w}}) > r(1+\epsilon) \text{ and } \exists l \in [1:L],\ H^{\mathcal{B}}_l(\check{\boldsymbol{x}}) = H^{\mathcal{B}}_l(\mathcal{P}_{\boldsymbol{w}}) \right\},$$

where $\epsilon > 0$ is the given small constant. Then the set cardinality $|\mathcal{S}| < cL$.

First, we prove that $\mathbf{F}_1$ **holds with probability at least** $1 - \frac{1}{e}$.

Let us consider the converse case that $H_l^{\mathcal{B}}(\boldsymbol{x}^*) \neq H_l^{\mathcal{B}}(\mathcal{P}_{\boldsymbol{w}})$ for $\forall l \in [1:L]$ whose probability is

$$\begin{aligned}
&\mathbf{Pr}\left[H_l^{\mathcal{B}}(\boldsymbol{x}^*) \neq H_l^{\mathcal{B}}(\mathcal{P}_{\boldsymbol{w}}), \forall l \in [1:L]\right] \\
=&\left(\mathbf{Pr}\left[H_l^{\mathcal{B}}(\boldsymbol{x}^*) \neq H_l^{\mathcal{B}}(\mathcal{P}_{\boldsymbol{w}})\right]\right)^L \\
=&\left(1-\mathbf{Pr}\left[H_l^{\mathcal{B}}(\boldsymbol{x}^*) = H_l^{\mathcal{B}}(\mathcal{P}_{\boldsymbol{w}})\right]\right)^L \\
\leq&\left(1-p_1^k\right)^L = \left(1-p_1^{\log_{\frac{1}{p_2}} n}\right)^{n^\rho} = \left(1-n^{-\rho}\right)^{n^\rho} \\
=&\left(\left(1-n^{-\rho}\right)^{-n^\rho}\right)^{-1} \leq \frac{1}{e}, \qquad (1)
\end{aligned}$$

where Inequality (1) follows from the inequality $(1-n^{-\rho})^{-n^\rho} \geq e$. Herewith we derive

$$\begin{aligned}
&\mathbf{Pr}\left[H_l^{\mathcal{B}}(\boldsymbol{x}^*) = H_l^{\mathcal{B}}(\mathcal{P}_{\boldsymbol{w}}), \exists l \in [1:L]\right] \\
=&1-\mathbf{Pr}\left[H_l^{\mathcal{B}}(\boldsymbol{x}^*) \neq H_l^{\mathcal{B}}(\mathcal{P}_{\boldsymbol{w}}), \forall l \in [1:L]\right] \\
\geq&1-\frac{1}{e}.
\end{aligned}$$

Second, we prove that $\mathbf{F}_2$ **holds with probability at least** $1-\frac{1}{c}$.

For every false alarm point $\check{\boldsymbol{x}}$ conforming to $\mathcal{D}(\check{\boldsymbol{x}}, \mathcal{P}_{\boldsymbol{w}}) > r(1+\epsilon)$, in any hash table $l \in [1:L]$ we have

$$\begin{aligned}
&\mathbf{Pr}\left[H_l^{\mathcal{B}}(\check{\boldsymbol{x}}) = H_l^{\mathcal{B}}(\mathcal{P}_{\boldsymbol{w}})\right] \\
<&(p_2)^k = (p_2)^{\log_{\frac{1}{p_2}} n} = \frac{1}{n}.
\end{aligned}$$

Therefore the expected number of false alarm points, which fall into the same hash bucket with the query $\mathcal{P}_{\boldsymbol{w}}$ in hash table l, is smaller than $n \times 1/n = 1$. Immediately, we conclude $\mathbf{E}\left[|\mathcal{S}|\right] < L$. Subsequently, we further apply Markov's inequality to derive the following result:

$$\mathbf{Pr}\left[|\mathcal{S}| \geq cL\right] \leq \frac{\mathbf{E}\left[|\mathcal{S}|\right]}{cL} < \frac{L}{cL} = \frac{1}{c},$$

which leads to

$$\mathbf{Pr}\left[|\mathcal{S}| < cL\right] = 1 - \mathbf{Pr}\left[|\mathcal{S}| \geq cL\right] > 1 - \frac{1}{c}.$$

Third, we prove that $\mathbf{F}_1$ **and** $\mathbf{F}_2$ **simultaneously hold with probability at least** $1-\frac{1}{c}-\frac{1}{e}$.

Let us deduce the conditional probability $\mathbf{Pr}\left[\overline{\mathbf{F}}_2|\mathbf{F}_1\right]$ as follows

$$\mathbf{Pr}\left[\overline{\mathbf{F}}_2|\mathbf{F}_1\right] = \frac{\mathbf{Pr}\left[\overline{\mathbf{F}}_2 \bigcap \mathbf{F}_1\right]}{\mathbf{Pr}\left[\mathbf{F}_1\right]} \leq \frac{\mathbf{Pr}\left[\overline{\mathbf{F}}_2\right]}{\mathbf{Pr}\left[\mathbf{F}_1\right]} = \frac{1-\mathbf{Pr}\left[\mathbf{F}_2\right]}{\mathbf{Pr}\left[\mathbf{F}_1\right]} < \frac{1-(1-\frac{1}{c})}{\mathbf{Pr}\left[\mathbf{F}_1\right]} = \frac{1}{c\mathbf{Pr}\left[\mathbf{F}_1\right]}.$$

Then, we derive

$$\begin{aligned}
\mathbf{Pr}\left[\mathbf{F}_1 \bigcap \mathbf{F}_2\right] &= \mathbf{Pr}\left[\mathbf{F}_2|\mathbf{F}_1\right]\mathbf{Pr}\left[\mathbf{F}_1\right] = (1-\mathbf{Pr}\left[\overline{\mathbf{F}}_2|\mathbf{F}_1\right])\mathbf{Pr}\left[\mathbf{F}_1\right] \\
&> (1-\frac{1}{c\mathbf{Pr}\left[\mathbf{F}_1\right]})\mathbf{Pr}\left[\mathbf{F}_1\right] = \mathbf{Pr}\left[\mathbf{F}_1\right] - \frac{1}{c} \\
&\geq 1-\frac{1}{c}-\frac{1}{e}.
\end{aligned}$$

To sum up, the inequality $\mathbf{Pr}\left[\mathbf{F}_1 \bigcap \mathbf{F}_2\right] > 1-\frac{1}{c}-\frac{1}{e}$ uncovers that with a constant probability larger than $1-\frac{1}{c}-\frac{1}{e}$, the BH-Hash algorithm is guaranteed to return at least a database point $\hat{\boldsymbol{x}}$ conforming to $\mathcal{D}(\hat{\boldsymbol{x}}, \mathcal{P}_{\boldsymbol{w}}) \leq r(1+\epsilon)$ by checking the first cL database points that collide to the query $\mathcal{P}_{\boldsymbol{w}}$ in either of the L hash tables. The algorithm terminates when cL database points have been scanned. Since at most L hash tables are visited, at most $L \cdot k = n^\rho \log_{1/p_2} n$ hash functions are evaluated. Accordingly, at most $cL = cn^\rho$ computations of the distance function $\mathcal{D}$ are needed to confirm the $(1+\epsilon)$-approximate nearest neighbors. Complete the proof. $\square$

Table 1. Results on **20 Newsgroups** using varying hash code length.

single hash table	8 bits			12 bits			16 bits		
Method	**MAP**	**Preprocess Time**	**Search Time**	**MAP**	**Preprocess Time**	**Search Time**	**MAP**	**Preprocess Time**	**Search Time**
AH-Hash	0.7142	0.07	0.07	0.7047	0.13	0.05	0.7074	0.17	0.05
EH-Hash	0.7588	383.0	3.41	0.7580	384.1	3.38	0.7346	385.2	3.33
BH-Hash	0.8140	0.06	0.16	0.7892	0.10	0.10	0.7752	0.14	0.07
LBH-Hash	0.8184	326.8	0.17	0.8162	506.7	0.10	0.8011	677.2	0.06
Random	0.6999	–	–	—					
Exhaustive	0.8426	–	0.52	—					

Table 2. Results on **Tiny-1M** using 8 and 12 hash bits.

single hash table	8 bits			12 bits		
Method	**MAP**	**Preprocess Time**	**Search Time**	**MAP**	**Preprocess Time**	**Search Time**
AH-Hash	0.2488	1.0	0.12	0.2417	1.4	0.10
EH-Hash	0.3182	577.2	0.98	0.3147	884.1	0.54
BH-Hash	0.3213	1.2	0.86	0.3208	1.4	0.58
LBH-Hash	0.3252	296.6	1.38	0.3313	419.4	0.98
Random	0.2440	–	–	—		
Exhaustive	0.3356	–	14.2	—		

Theorem 2 indicates that the query time of the BH-Hash algorithm is essentially bounded by $O(n^{\rho})$, in which the exponent $0 < \rho < 1$ relies on both r and ϵ. In fact, it can be simplified under additional assumptions. We present the following Corollary.

Corollary 1. If $r \geq \gamma\pi^2/4$ with $0 < \gamma < 1$, then the query time of the BH-Hash algorithm is bounded by $O(n^{\frac{\ln(2/(1-\gamma))}{\ln 2+\gamma(1+\epsilon)}})$.

Proof. Given a fixed $\epsilon > 0$,

$$\rho = \frac{\ln p_1}{\ln p_2} = \frac{\ln\left(\frac{1}{2} - \frac{2r}{\pi^2}\right)}{\ln\left(\frac{1}{2} - \frac{2r(1+\epsilon)}{\pi^2}\right)},$$

which can be proved to be a monotonically decreasing function with respect to the variable r. Then, if $r \geq \gamma\pi^2/4$,

$$\rho \leq \frac{\ln\left(\frac{1}{2} - \frac{\gamma}{2}\right)}{\ln\left(\frac{1}{2} - \frac{\gamma(1+\epsilon)}{2}\right)} = \frac{\ln\left(\frac{1-\gamma}{2}\right)}{\ln\left(1-\gamma(1+\epsilon)\right) - \ln 2} \leq \frac{\ln\left(\frac{1-\gamma}{2}\right)}{-\gamma(1+\epsilon) - \ln 2} = \frac{\ln\left(\frac{2}{1-\gamma}\right)}{\ln 2 + \gamma(1+\epsilon)},$$

where we use the inequality $\ln\left(1-\gamma(1+\epsilon)\right) \leq -\gamma(1+\epsilon)$.

Thus the query time bound $O(n^{\rho})$ does not exceed $O\left(n^{\frac{\ln(2/(1-\gamma))}{\ln 2+\gamma(1+\epsilon)}}\right)$. It completes the proof. □

2. Experimental Results

Tables 1-3, which again corroborate both accuracy and speed advantages of LBH-Hash, are what we have mentioned in the main paper. We keep a single hash table and adopt varying hash code length, ranging from 8 bits to 20 bits, for each of four compared hyperplane hashing methods. To perform hash lookups, we set the Hamming radius to 1, 2, 3, and 4 for 8, 12, 16, and 20 hash bits, respectively. In Tables 1-3, mean average precision (MAP) ($[0, 1]$) is the mean of the SVM's average precision after 300 active learning iterations over all classes and 5 runs; "Preprocess Time" (seconds) refers to the preprocessing time for a hashing method to

Table 3. Results on **Tiny-1M** using 16 and 20 hash bits.

single hash table	16 bits			20 bits		
Method	**MAP**	**Preprocess Time**	**Search Time**	**MAP**	**Preprocess Time**	**Search Time**
AH-Hash	0.2404	1.6	0.10	0.2444	1.7	0.09
EH-Hash	0.3094	1058.1	0.22	0.3075	1343.6	0.15
BH-Hash	0.3141	1.8	0.16	0.3010	2.0	0.12
LBH-Hash	0.3341	586.5	0.95	0.3469	883.7	0.88
Random	0.2440	–	–	—		
Exhaustive	0.3356	–	14.2	—		

compress all of the database points to hash codes (for LBH-Hash, such time includes the time spent on learning the hash functions from the training data); "Search Time" (seconds) refers to the average search time per query. Unlike the other methods, AH-Hash always uses twice hash bits to follow its dual-bit hashing theme.

Let us consider the reported MAP first. Table 1 reveals that Random<AH-Hash<EH-Hash<BH-Hash<LBH-Hash<Exhaustive (where < indicates inferior MAP) on **20 Newsgroups**. On **Tiny-1M**, Tables 2 and 3 show that Random≈AH-Hash<EH-Hash<BH-Hash<LBH-Hash<Exhaustive when using bits shorter than 20, and that Random≈AH-Hash<BH-Hash<EH-Hash<Exhaustive<LBH-Hash when using 20 bits. LBH-Hash consistently surpasses the other hashing methods in terms of MAP.

Second, it is observed that AH-Hash and BH-Hash are both efficient in terms of preprocessing time, while EH-Hash and LBH-Hash need much longer preprocessing time. The preprocessing time of EH-Hash is $O(d^2n + dkn)$ due to the $O(d^2 + dk)$-complexity EH hash function computation for each database point. The preprocessing time of LBH-Hash is $O(2dkn + (dm + m^2)Tk)$ in which m ($\ll n$) is the number of sampled training points and T is the number of optimization iterations. It is noted that EH-Hash's time is quadratic in the data dimension d ($k \ll d$) while LBH-Hash's time is linear in d. For those really large-scale datasets with $O(10^3)$ or even higher dimension, the quadratic dimension dependence of EH-Hash will trigger unaffordable computational costs, while our proposed hashing schemes including both BH-Hash and LBH-Hash will enjoy the linear dependence on dimension. For example, EH-Hash's preprocessing time is about twice longer than LBH-Hash's on the **Tiny-1M** dataset.

Although LBH-Hash takes longer preprocessing time than BH-Hash as shown in Tables 1-3, the reported time is mostly spent on offline training and indexing. In practice, the online search (query) time is more critical. Considering the search time, all of the compared hashing methods except EH-Hash are much faster than the exhaustive search. Although AH-Hash is fastest, it incurs many empty hash lookups, as disclosed in Figs. 3(c) and 4(c) of the main paper. On **20 Newsgroups** that is very high-dimensional (over 20K), EH-Hash is slowest due to the above-mentioned high complexity of its hash function computation, and even slower than exhaustive search. On **20 Newsgroups**, BH-Hash and LBH-Hash demonstrate almost the same fast search speed; on **Tiny-1M**, LBH-Hash is slower than BH-Hash, but is acceptably fast for online search.

References

Gionis, A., Indyk, P., and Motwani, R. Similarity search in high dimensions via hashing. In *Proc. VLDB*, 1999.