International Journal of Innovative
Computing, Information and Control
Volume **16**, Number **3**, June **2020**

ICIC International ©2020 ISSN 1349-4198
pp. 1125–1132

APPROXIMATE CONSISTENT WEIGHTED SAMPLING FOR EFFICIENT TOP-K SEARCH

Yunna Kim and Heasoo Hwang*

Department of Computer Science and Engineering
University of Seoul
163 Seoulsiripdaero, Dongdaemun-gu, Seoul 02504, Korea
ynkim31@uos.ac.kr; *Corresponding author: hwang@uos.ac.kr

Received November 2019; revised February 2020

Abstract. *Top-k search on weighted sets can be very slow since the computation cost of generalized Jaccard similarity is proportional to the dimensionality of sets. ICWS generates samples of high quality, but its hashing cost is too high to generate samples from high-dimensional weighted sets. We propose simple hashing methods, ICWS_P and its variants, that approximate ICWS very efficiently in $O\left(D \cdot \frac{K}{B}\right)$. Extensive experiments show that hashing cost is reduced significantly while top-k precision and classification accuracy with estimated set similarity are almost as high as those of ICWS. Query time can also be improved since less than K samples are compared for sets of low similarity.*
Keywords: Generalized Jaccard similarity, Minwise hashing, Weighted sampling

1. **Introduction.** The amount and the dimensionality of data such as images, textual contents and speech recordings are growing very rapidly due to the wide spread availability of high-resolution recording and storage. For instance, a news article can be represented as a high-dimensional vector of term occurrences. Its dimension further increases if it encodes co-occurrences of terms such as bigrams and trigrams. Compressing such high-dimensional data into compact representation becomes critical for efficient top-k search.

To handle high-dimensional data, many hashing algorithms including minwise hashing have been suggested. For a given binary vector, minwise hashing generates K hashes that satisfies LSH (Locality Sensitive Hashing) property, i.e., the hash collision probability is equal to Jaccard similarity. However, it takes O(DK) to generate K hashes from a binary vector with D nonzero values. Since K is usually set to hundreds or thousands [1], hash generation can be very slow for high-dimensional binary vectors. OPH (One Permutation Hashing) [2] successfully reduces expensive K permutations of minwise hashing to one permutation while generating K hashes that still satisfy LSH property. The work of [1,3,4] improves OPH by densifying sparse vectors, achieving O(D + K). However, this does not apply to non-binary vectors, i.e., weighted sets.

WMH (Weighted Minwise Hashing) generates hashes for weighted sets. Let us consider two weighted sets, $\mathbf{S} = (S_k)$ and $\mathbf{T} = (T_k)$ where S_k and T_k represent the weight of the kth element of $\mathbf{S}$ and $\mathbf{T}$ in the universal set $\mathbf{U}$. The generalized Jaccard similarity, $\mathrm{GJS}(\mathbf{S}, \mathbf{T})$, is defined as $\frac{\sum_k \min(S_k, T_k)}{\sum_k \max(S_k, T_k)}$. CWS (Consistent Weight Sampling) [5] generates a sample, (k, y_k): $0 < y_k \leq S_k$, such that the sample collision probability is equal to $\mathrm{GJS}(\mathbf{S}, \mathbf{T})$. It satisfies two important properties: (1) **uniformity**: a sample (k, y_k) should be uniformly sampled from $\bigcup_k (\{k\} \times [0, S_k])$, i.e., the probability of selecting k is proportional to S_k, and y_k is uniformly distributed in $[0, S_k]$; (2) **consistency**: given two non-empty set $\mathbf{S}$

DOI: 10.24507/ijicic.16.03.1125

and $\mathbf{T}$, if $\forall k$, $0 < T_k \leq S_k$, a sample (k, y_k) is generated from $\mathbf{S}$ and satisfies $y_k \leq T_k$, then the sample (k, y_k) will be generated from $\mathbf{T}$ as well. ICWS [6] is a popular state-of-the-art CWS method that addresses the efficiency issue of CWS. As described in Algorithm 1, for every nonzero element S_k, it computes a_k using three random numbers to generate a sample (k^*, y_{k^*}). Therefore, its hashing cost is O(DK) for K samples, just as that of minwise hashing. When D and K are large, its hashing time can be extremely long.

1. For all k $(S_k > 0)$ do:
   (1) $r_k \sim Gamma(2,1)$, $c_k \sim Gamma(2,1)$, $\beta_k \sim Uniform(0,1)$
   (2) $t_k = \left\lfloor \frac{\ln S_k}{r_k} + \beta_k \right\rfloor$, $y_k = \exp(r_k(t_k - \beta_k))$, $z_k = y_k \exp(r_k)$, $a_k = \frac{c_k}{z_k}$
2. Find $k^* = \arg\min_k a_k$ and generate sample (k^*, y_{k^*})

Algorithm 1. Pseudocode of ICWS [6]

In this paper, we target on generating WMH samples efficiently for high-dimensional weighted sets in order to perform top-k search or k-NN (k-Nearest Neighbor) classification. To do so, we propose ICWS_P (ICWS Proportional), a simple and efficient hashing method that approximates CWS samples. By partitioning $\mathbf{U}$ into bins and generating samples proportionally to the weight sum of each bin, it reduces hashing time while maintaining sample quality. Given B, the number of bin, its hashing cost is $\mathrm{O}\big(\mathrm{D} \cdot \frac{\mathrm{K}}{\mathrm{B}}\big)$ while requiring no densification of empty bins. In addition, during query time, it enables us to estimate GJS between sets using less samples than K especially when two sets are of low similarity.

This paper is organized as follows. We review important prior works on the efficiency of CWS in Section 2 and propose our methods in Section 3. Then, we present extensive experimental results in Section 4 and conclude in Section 5.

2. **Related Work.** We briefly review prior works on the efficiency of WMH. ICWS [6] improves the efficiency of the original CWS [5] by requiring O(DK) hashing time to generate K samples from a weighted set with D nonzero weights. As a simplification of CWS, 0-bit CWS [7] is proposed. Motivated by the observation that $P[(k, y_k) = (k', y_{k'})] \approx P(k = k')$ for (k, y_k) from $\mathbf{S}$ and $(k', y_{k'})$ from $\mathbf{T}$, 0-bit CWS uses only k and k' as samples. The space efficiency and GJS estimation time are improved, and hashing time still remains similar to ICWS. Red-green map [8] is not based on CWS, but we want to mention it here since it sometimes generates WMH samples extremely faster than ICWS. However, it can be applied only if the maximum weight of each dimension is known in advance, which is usually impossible in real-world scenarios.

CCWS (Canonical CWS) [9] is motivated by the fact that ICWS is based on implicit quantization on the logarithm of S_k ($\ln S_k$), instead of S_k. This logarithmic transformation results in unevenly quantized subelements and thus may map different active subelements into the same subelements, increasing the collision probability. To restore the uniformity property of ICWS, CCWS replaces $\ln S_k$ with S_k, and this removal of logarithmic transformation leads to slight improvement in the actual hashing time. PCWS (Practical CWS) [10] transforms ICWS formulas into equivalent ones and simplifies them by using one less uniform random variable and replacing the formula for a_k in Algorithm 1 with a less complicated one. This simple modification leads to 20% less memory usage during hashing and 1/5∼1/3 reduction of hashing time. I^{2}CWS (Improved ICWS) [11] fixes the dependence of k and y_k in sample (k, y_k) by sampling y_k and z_k separately rather than deriving z_k from y_k. The approximation accuracies on diverse synthetic datasets are better than ICWS, but classification accuracy and top-k precision on real datasets are similar to

ICWS while spending more hashing time. Overall, in [9-11], the time complexity of hash generation is still the same as ICWS, O(DK).

SCWS [12] improves the efficiency of 0-bit CWS [7] by eliminating the floor function in t_k in Algorithm 1 and simplifies the formula for a_k as a floating-point multiplication using a pool of pre-sampled values. ICWS_B [13] is a simple hashing method that applies the partitioning idea of OPH [2] to reducing its hashing cost to $\mathrm{O}\big(\mathrm{D} \cdot \frac{\mathrm{K}}{\mathrm{B}}\big)$ using B bins. While OPH partitions a set into K bins and selects a sample per bin, ICWS_B partitions a weighted set into B bins and selects $\frac{\mathrm{K}}{\mathrm{B}}$ samples per bin. While the hashing time is reduced, the sample quality is not as good as that of ICWS. This is mainly due to the fact that uniformity holds only within each bin, not in the entire set and large B introduces many empty bins from which dummy samples are generated. To address the empty bin issue, ICWS_D [13] adds densification to ICWS_B, but the low sample quality does not improve significantly.

3. **Our Methods.** We propose simple hashing methods for weighted sets, ICWS_P and its two variants (ICWS_P1 and ICWS_P2), that approximate ICWS very efficiently by generating samples per bin. Our methods are different from existing bin-based approaches such as OPH and ICWS_B in that the number of samples per bin is not fixed but determined proportionally to the sum of weights in a bin. Notice that no sample is generated from empty bins, and thus densification is unnecessary while other bin-based approaches need to handle empty bins to improve sample quality.

ICWS_P generates B bins by partitioning $\mathbf{U}$ into $\frac{\mathrm{D}}{\mathrm{B}}$ consecutive dimensions originally defined by the dataset. Then, it generates $\mathrm{N}(b_i)$ samples from bin b_i based on the sum of nonzero weights in b_i, $\mathrm{W}(b_i)$, by following ICWS [6] scheme as in lines 2-3 of Algorithm 2. This can be replaced by any accurate CWS method such as I^{2}CWS [11]. Since each sample (k^*, y_{k^*}) from b_i satisfies $k^* = \arg\min_{l \in b_i} a_l$, bin construction strategy may affect sample quality from bins. In fact, ICWS_P approximates ICWS effectively, but we observed that its performance sometimes degrades when we increase B to speed up ICWS_P as we can see from the result on cifar10 dataset in Figure 1.

Input (1) $\mathbf{S}$: a weighted set; (2) K: number of samples; (3) B: number of bins
1. Partition $\mathbf{U}$ into B bins
2. Calculate the weight sum of each bin b_i, $\mathrm{W}(b_i)$, and determine the number of samples for b_i, $\mathrm{N}(b_i) = \mathrm{K} * \frac{\mathrm{W}(b_i)}{\sum_j \mathrm{W}(b_j)}$
3. For each b_i, generate $\mathrm{N}(b_i)$ samples using ICWS by considering b_i as a weighted set
   - For all k ($S_k > 0$) do:
     (1) $l = k \ \% \ \frac{\mathrm{D}}{\mathrm{B}}$
     (2) $r_l \sim Gamma(2,1)$, $c_l \sim Gamma(2,1)$, $\beta_l \sim Uniform(0,1)$
     (3) $t_l = \left\lfloor \frac{\ln S_k}{r_l} + \beta_l \right\rfloor$, $y_l = \exp(r_l(t_l - \beta_l))$, $z_l = y_l \exp(r_l)$, $a_l = \frac{c_l}{z_l}$
   - Find $k^* = \arg\min_{l \in b_i} a_l$ and generate sample (k^*, y_{k^*})
4. Output K samples generated in Step 3 with $[\mathrm{N}(b_1), \mathrm{N}(b_2), \ldots, \mathrm{N}(b_\mathrm{B})]$

Algorithm 2. Pseudocode of our methods

Therefore, we suggest two variants of ICWS_P, ICWS_P1 and ICWS_P2, that eliminate or reduce the impact of dependent dimensions by using different bin construction strategies. ICWS_P1 generates bins by randomly permuting $\mathbf{U}$ using a linear congruential hash function $(p_1 k + p_2) \bmod \mathrm{B}$, where p_1 and p_2 are large prime numbers such that $p_1 \bmod 4 = 1$ and $p_2 \bmod 2 = 1$. ICWS_P2 integrates ICWS_P and ICWS_P1 by selecting

$\frac{\mathrm{K}}{2}$ samples from bins with consecutive dimensions as ICWS_P and the rest from those with randomly permuted dimensions as ICWS_P1.

Our methods satisfy uniformity and consistency properties of ICWS. Let $\mathrm{W}(\mathbf{S})$ denote the total weight of $\mathbf{S}$. The probability of our methods selecting k^* is as follows. Let us assume that dimension k^* belongs to bin b_i after bin construction. We select a sample from b_i with probability $\frac{\mathrm{W}(b_i)}{\sum_j \mathrm{W}(b_j)} = \frac{\mathrm{W}(b_i)}{\mathrm{W}(\mathbf{S})}$. Within b_i, since we select samples with ICWS, uniformity is guaranteed within each bin, which means that k^* is selected with probability $\frac{S_{k^*}}{\mathrm{W}(b_i)}$. Therefore, we select k^* with probability $\frac{\mathrm{W}(b_i)}{\mathrm{W}(\mathbf{S})} \cdot \frac{S_{k^*}}{\mathrm{W}(b_i)}$, which is equal to $\frac{S_{k^*}}{\mathrm{W}(\mathbf{S})}$, satisfying the uniformity of ICWS. Next, our methods satisfy the consistency of ICWS as well. Let us assume that $\mathbf{S}$ dominates $\mathbf{T}$, i.e., $\forall k$, $0 < T_k \leq S_k$, and (k^*, y_{k^*}) with $y_{k^*} \leq T_{k^*}$ is sampled from $\mathbf{S}$. We allow (k^*, y_{k^*}) to be sampled only from the bin containing k^*, i.e., b_i. Since we use ICWS for sample generation from each bin, if (k^*, y_{k^*}) is sampled from b_i of $\mathbf{S}$, (k^*, y_{k^*}) will also be sampled from b_i of $\mathbf{T}$ by the consistency of ICWS.

During query time, we count the number of sample collisions among K samples of $\mathbf{S}$ and $\mathbf{T}$ in order to estimate $\mathrm{GJS}(\mathbf{S}, \mathbf{T})$. Notice that $\mathrm{N}(b_i)$ of $\mathbf{S}$, $\mathrm{N_S}(b_i)$, and $\mathrm{N}(b_i)$ of $\mathbf{T}$, $\mathrm{N_T}(b_i)$, may not be the same. This is related to the uniformity of our methods; since we want to sample a particular dimension k^* in b_i with probability $\frac{S_{k^*}}{\mathrm{W}(\mathbf{S})}$, the number of samples from b_i, $\mathrm{N_S}(b_i)$, is set to $\frac{\sum_{k^* \in b_i} S_{k^*}}{\mathrm{W}(\mathbf{S})} \cdot \mathrm{K} = \frac{\mathrm{W}(b_i)}{\mathrm{W}(\mathbf{S})}\mathrm{K}$. Assume that $\mathrm{N_S}(b_i) < \mathrm{N_T}(b_i)$. Then, we compare only $\mathrm{N_S}(b_i)$ samples of $\mathbf{S}$ and $\mathbf{T}$. The following $(\mathrm{N_T}(b_i) - \mathrm{N_S}(b_i))$ samples of $\mathbf{T}$ do not need to be compared since it is certain that there are $(\mathrm{N_T}(b_i) - \mathrm{N_S}(b_i))$ samples of $\mathbf{S}$ that are sampled from $b_{j(j \neq i)}$, so collision cannot happen. Therefore, whenever an empty bin is found in $\mathbf{S}$, we can skip all the samples from the corresponding bin of $\mathbf{T}$. In contrast, we compare all the K samples if $\forall i$, $\frac{\mathrm{W}(b_i)}{\sum_j \mathrm{W}(b_j)}$ of $\mathbf{S}$ and $\frac{\mathrm{W}(b_i)}{\sum_j \mathrm{W}(b_j)}$ of $\mathbf{T}$ are the same, which usually happens when $\mathbf{S}$ and $\mathbf{T}$ are highly similar. For this, we need to keep $[\mathrm{N}(b_1), \mathrm{N}(b_2), \ldots, \mathrm{N}(b_\mathrm{B})]$ with K samples as in line 4 of Algorithm 2, which require $\mathrm{B} \cdot \lceil \log_2 \mathrm{B} \rceil$ bits. By comparing less samples between sets of low similarity, the average number of comparisons for GJS estimation can be significantly reduced. For instance, given $\mathrm{K} = 1{,}024$ and $\mathrm{B} = 32$, ICWS_P used only 574 comparisons on average on protein dataset. As will be seen in Figure 2 and Figure 3, our precision@20 and classification accuracy are better than ICWS that always performs 1,024 comparisons.

4. **Experimental Results.** We use 6 real datasets with various dimensions and densities in LIBSVM [14] as shown in Table 1. From each dataset, we randomly select 6,000 sets and use 1,000 sets as queries. For datasets with negative values, we add a large integer to each non-zero value to remove negative values while maintaining density. On a dataset, we execute each sampling method ten times with various combinations of parameters to measure hashing time, top-k precision and classification accuracy.

TABLE 1. Characteristics of datasets and hashing time (sec) (K = 1,024)

Dataset	Dimension	Density	ICWS [4]	PCWS [8]	**ICWS_P** (B = 32)
gisette	5,000	0.99	1,501.5	1,481.6(98.7%)	60.4(4.0%)
cifar10	3,072	0.99	965.5	941.1(97.5%)	33.4(3.5%)
rcv1.binary	47,236	0.0015	102.2	96.4(94.3%)	7.6(7.4%)
mnist	780	0.18	50.0	47.6(95.2%)	3.7(7.4%)
satimage	36	0.99	14.1	13.3(94.3%)	1.2(8.5%)
protein	357	0.28	41.1	38.8(94.4%)	2.6(6.3%)

We compare the performance of our methods, ICWS_P and its two variants (ICWS_P1 and ICWS_P2), with three existing methods, ICWS [6], PCWS [10] and ICWS_B [13]. We exclude 0-bit CWS [7] and ICWS_D [13] since we observed that 0-bit CWS showed almost exactly the same performance as ICWS and ICWS_D performed worse than ICWS_B on all the datasets.

Table 1 shows the dimension and the average density of each of 6 datasets and hashing time required by ICWS, PCWS and ICWS_P, our method. Firstly, ICWS spends 1,501.5 secs to hash gisette dataset. While PCWS has been proposed to simplify ICWS to reduce hashing time, the reduction rate is shown to be less than 6% on our datasets. In contrast, ICWS_P requires much less hashing time than ICWS or PCWS on all the datasets. It spends only 60.4 secs to hash gisette, which is only 4% of the hashing time by ICWS. Furthermore, as shown in Table 2, if we increase the number of bins (B), bin-based approaches including our methods can hash weighted sets very efficiently. While ICWS spends 965.5 secs to hash cifar10 dataset, ICWS_P can process it using 7.3 secs, only 0.8% of the hashing time by ICWS. Recall that the time complexity of our methods is $O\big(D \cdot \frac{K}{B}\big)$ and that of ICWS is O(DK).

TABLE 2. Hashing time (sec) with different number of bins (cifar10, K = 1,024)

ICWS	PCWS	B	ICWS_B	**ICWS_P**	**ICWS_P1**	**ICWS_P2**
965.5	941.1(97.5%)	8	134.7(13.9%)	125.6(13.0%)	125.4(13.0%)	128.2(13.3%)
		32	34.4(3.6%)	33.4(3.5%)	36.3(3.8%)	37.8(3.9%)
		128	9.0(0.9%)	11.6(1.2%)	13.3(1.4%)	14.9(1.5%)
		512	2.6(0.3%)	7.3(0.8%)	8.5(0.9%)	12.0(1.2%)

Next, we want to compare the quality of samples by estimating top-k precision with various k's and the accuracy of 1NN classification. We do not present MSE since our goal is not to estimate all the GJS scores accurately but to perform effective top-k search using samples that are efficiently generated from large-scale high-dimensional datasets. Experiments show that MSE of ICWS_P is not as low as that of ICWS, but low enough to perform top-k search and 1NN classification effectively.

Figure 1 shows precision@50 of 6 methods (upper charts) and classification accuracy of 7 methods including GJS (lower charts) on three datasets of high density or high dimension. While ICWS_P requires much less hashing time than ICWS as shown in Table 2, its precision@50 and classification accuracy are almost as high as those of ICWS on gisette and rcv1.binary. On cifar10, precision@50 gradually drops as B increases, but it is restored by using different bin construction strategies of ICWS_P1 or ICWS_P2. For instance, if B = 128, precision@50 and classification accuracy of ICWS_P1 or ICWS_P2 are almost as high as those of ICWS while using only about 1.5% of the ICWS hashing time.

In order to examine how the sample quality changes as the sample size K grows, we compare hashing time on cifar10 in Table 3 and precision@20 and classification accuracy on 6 datasets in Figure 2 and Figure 3 respectively.

As shown in Table 3, we can see that hashing time increases as more samples are generated. However, when K is fixed and B = 32, our methods spend less than 4% $\big(\approx \frac{1}{32}\big)$ of the ICWS hashing time. As shown in Table 2, hashing time of our methods can be further reduced to about 1% if larger B is used.

Figure 2 illustrates that ICWS_P performs as good as ICWS in terms of precision@20 on gisette, rcv1.binary, mnist and satimage. On cifar10, it is worse than ICWS as mentioned before, but ICWS_P1 and ICWS_P2 achieve higher precision@20 instead. On protein,

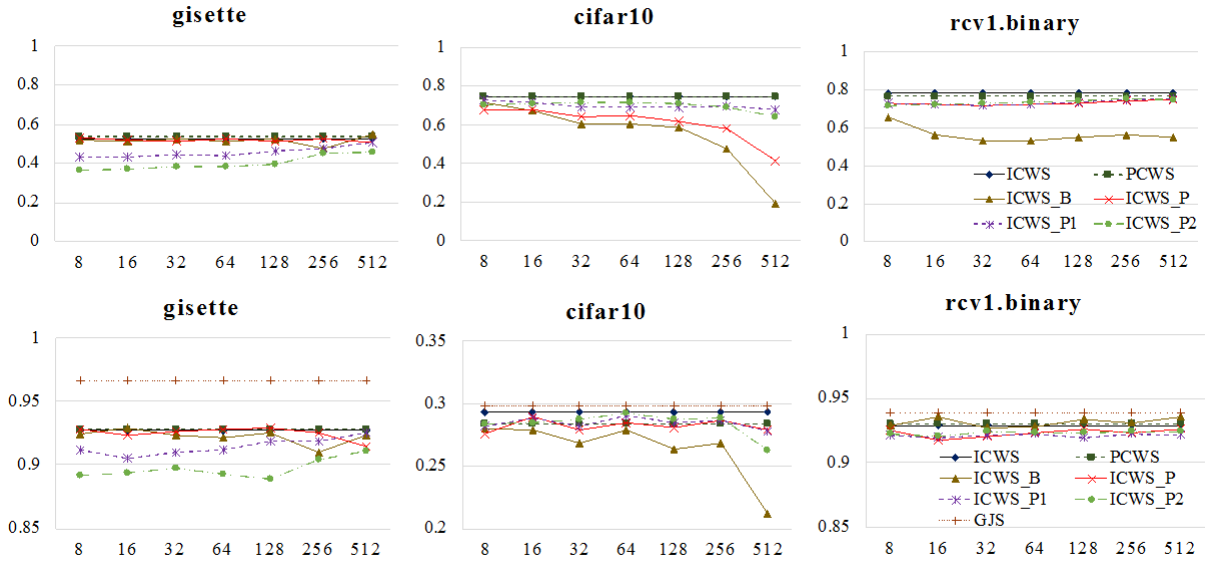

FIGURE 1. Prec@50 (upper) and classification accuracy (lower) as B changes (K = 1,024)

TABLE 3. Comparison of hashing time (sec) when generating K samples (cifar10, B = 32)

K	ICWS	PCWS	ICWS_B	**ICWS_P**	**ICWS_P1**	**ICWS_P2**
64	59.0	57.7(97.8%)	2.3(3.9%)	3.3(5.6%)	4.1(6.9%)	5.3(9.0%)
256	235.6	229.8(97.5%)	8.9(3.8%)	9.1(3.9%)	10.3(4.4%)	11.4(4.8%)
1,024	965.5	941.1(97.5%)	37.1(3.8%)	32.6(3.4%)	35.4(3.7%)	36.4(3.8%)

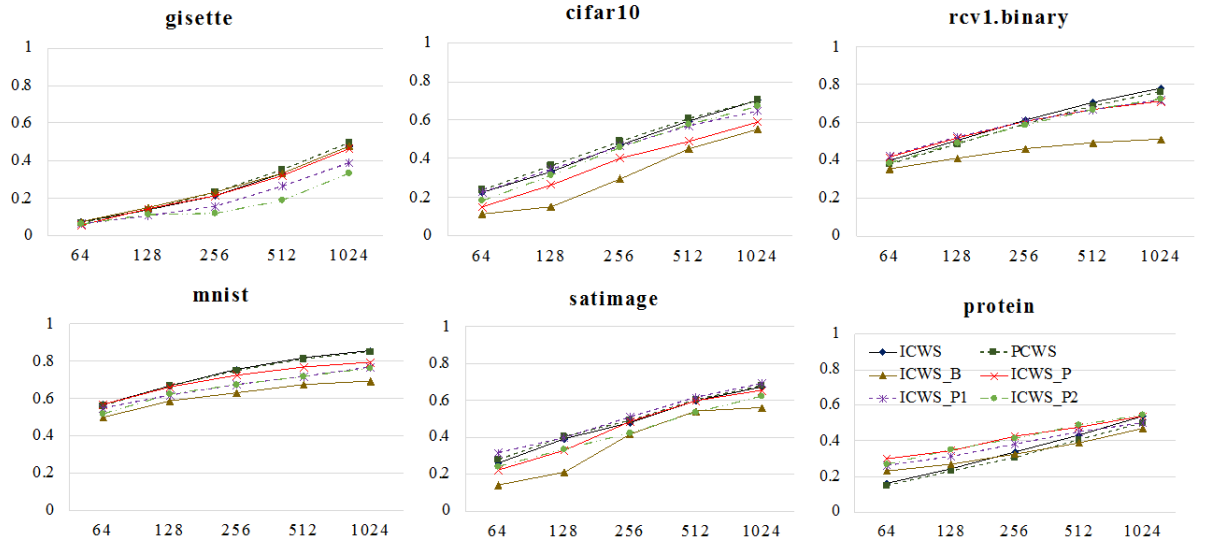

FIGURE 2. Precision@20 with K samples (B = 32)

our methods show the best precision@20. In Figure 3, ICWS_P shows similar classification accuracy as ICWS in the upper three charts. ICWS shows the highest classification accuracy except for GJS on mnist and protein. On satimage, the classification accuracy

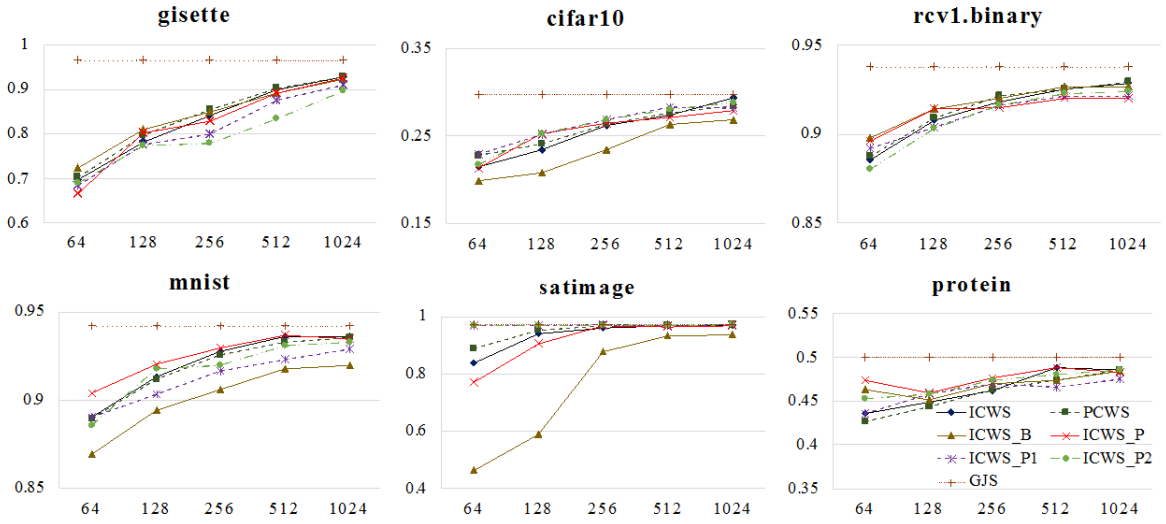

FIGURE 3. Classification accuracy with K samples (B = 32)

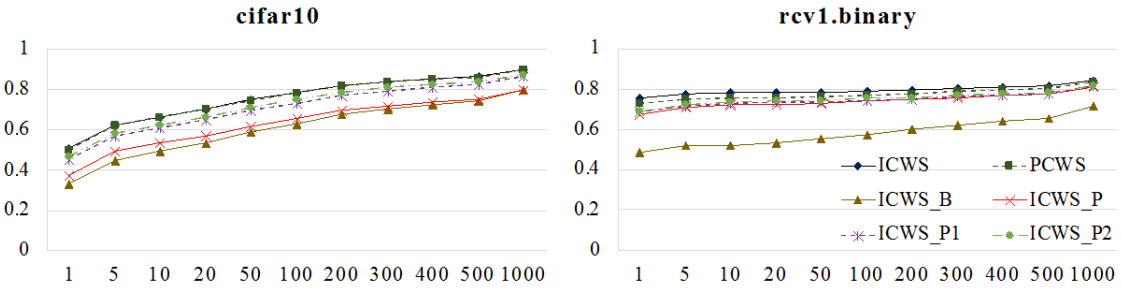

FIGURE 4. Precision@k with various k's (K = 1,024, B = 128)

of ICWS_P2 is as high as that of GJS. PCWS shows very similar precision@20 and classification accuracy as ICWS. ICWS_B shows similar performance as other methods on gisette and protein, but its precision@20 or classification accuracy is inferior to those of other methods on the other datasets.

Figure 4 shows top-k precision of 6 methods with various k's. K and B are fixed to 1,024 and 128 respectively. Recall that on cifar10, ICWS_P1 and ICWS_P2 show higher precision@50 than ICWS_P with various B's in Figure 1. In fact, their precision@50 scores are almost as good as that of ICWS. In addition, Figure 4 illustrates that ICWS_P1 and ICWS_P2 show almost the same precision@k as ICWS across various k's ranging from 1 to 1,000 even though they spend only 1.5% of the ICWS hashing time. On rcv1.binary, all of our methods perform as good as ICWS.

5. **Conclusions.** In order to perform fast hash generation of large-scale high-dimensional weighted sets for effective top-k search, we propose simple WMH methods, ICWS_P and its variants, which approximate ICWS very efficiently in $O\left(D \cdot \frac{K}{B}\right)$. Through extensive experiments, it is observed that as B increases, our hashing time decreases while our sample quality remains almost as good as that of ICWS on test datasets, even though ICWS spends much higher hashing cost than our methods do. In addition, query time can also be improved since our methods do not compare all the K samples when estimating GJS between sets of low similarity. As future work, we plan to enhance our methods

such that it can adjust B and bin construction strategy so that we can efficiently generate samples for highly effective top-k search on any given dataset.

Acknowledgment. This work was supported by the 2016 Research Fund of the University of Seoul. The authors gratefully acknowledge the helpful comments and suggestions of the reviewers, which have improved the presentation.

REFERENCES

[1] A. Shrivastava, Optimal densification for fast and accurate minwise hashing, *ICML*, pp.3153-3163, 2017.
[2] P. Li, A. Owen and C.-H. Zhang, One permutation hashing, *NIPS*, pp.3122-3130, 2012.
[3] A. Shrivastava and P. Li, Densifying one permutation hashing via rotation for fast near neighbor search, *ICML*, pp.557-565, 2014.
[4] A. Shrivastava and P. Li, Improved densification of one permutation hashing, *UAI*, 2014.
[5] M. Manasse, F. McSherry and K. Talwar, *Consistent Weighted Hashing*, Technical Report MSR TR-2010-73, 2010.
[6] S. Ioffe, Improved consistent sampling, weighted Minhash and L1 sketching, *ICDM*, pp.246-255, 2010.
[7] P. Li, 0-bit consistent weighted sampling, *KDD*, pp.665-674, 2015.
[8] A. Shrivastava, Simple and efficient weighted minwise hashing, *NIPS*, pp.1498-1506, 2016.
[9] W. Wu, B. Li, L. Chen and C. Zhang, Canonical consistent weighted sampling for real-value weighted Min-Hash, *ICDM*, pp.1287-1292, 2016.
[10] W. Wu, B. Li, L. Chen and C. Zhang, Consistent weighted sampling made more practical, *WWW*, pp.1035-1043, 2017.
[11] W. Wu, B. Li, L. Chen, C. Zhang and P. S. Yu, Improved consistent weighted sampling revisited, *TKDE*, doi: 10.1109/TKDE.2018.2876250, 2018.
[12] E. Raff, J. Sylvester and C. Nicholas, Engineering a simplified 0-bit consistent weighted sampling, *CIKM*, pp.665-674, 2018.
[13] Y. Kim and H. Hwang, A method on improving the efficiency of sampling for weighted sets, *KSC*, 2019.
[14] C.-C. Chang and C.-J. Lin, LIBSVM: A library for support vector machines, *TIST*, https://www.csie.ntu.edu.tw/∼cjlin/libsvm/, 2011.