Local Density Estimation in High Dimensions
Supplementary Materials

Xian Wu Moses Charikar Vishnu Natchu

Preliminaries

We introduce here some definitions and notation that will be used throughout this supplement.

Suppose we are given a set of K hash tables, with binary addresses of length t. Therefore, each hash table contains 2^t hash buckets with addresses in $\{0,1\}^t$. Let $d(i,j)$ denote the hamming distance between hash addresses i and j. Given a hash table, let B denote the set of buckets in the table, and let b_i be the number of elements in bucket i.

For any bucket address i, let $\mathcal{N}_i$ denote the set of neighbors of i, so address $j \in \mathcal{N}_i$ implies that $d(i,j) = 1$. For $a \in \{0,\dots,t\}$, let $\mathcal{N}_i(a)$ denote the neighbors of i which match i's hash sequence in the first a bits. Therefore, $j \in \mathcal{N}_i(a)$ implies that $d(i,j) = 1$ and the first a bits of i match the first a bits of j. Note that $\mathcal{N}_i(0)$ degenerates to $\mathcal{N}_i$ as a special case.

Let q denote the address of the bucket that a given query hashes to in a table. Oftentimes we are interested in elements that fall into buckets at certain hamming distances to q. Denote the set of hamming distances of interest $\mathcal{I}$. Without loss of generality we always assume $q = \mathbf{0}_t$, the hash address of all 0's.

Definition .1 (Counts Vector)**.** For a table k, a fixed hash address i, and $r = 0,1,\dots t$, the counts vector $C_i^k[r]$ gives the number of elements hashed to buckets at hamming distance r away from i.

Definition .2 (Counts Matrix)**.** For a table k and a fixed hash address i, the counts matrix $M_i^k \in \mathbb{Z}^{(t+1)\times(t+1)}$ has entries $M_i^k[s,a]$ that each gives a count of the number of elements hashed to buckets whose addresses are at hamming distance s away from i and share the same first a bits as address i.

A. Computing the Average Counts Vector

We propose a message-passing algorithm, `Aggregate-Counts` (Algorithm 1), which for a single table k computes C_i^k for all buckets i in time $O(t^2 2^t)$. Repeating for each of the K tables yields a total $O(Kt^2 2^t)$ runtime. `Aggregate-Counts` proceeds in t rounds where in each round r, each bucket i updates $C_i^k[r]$ by aggregating information passed from its neighbors in the set $\mathcal{N}_i$.

Theorem 3.1 (`Aggregate-Counts`)**.** Given a set of K hash tables, each with 2^t hash buckets with addresses in $\{0,1\}^t$, `Aggregate-Counts` (Algorithm 1) computes, for each hash address i, the number of elements in buckets that are hamming distance $0,1,\dots t$ away from i, in each of the K tables, in time $O(Kt^2 2^t)$.

A.1. Analysis

We first analyze the correctness of `Aggregate-Counts` (Algorithm 1). Since the algorithm proceeds in rounds, we want to show that at the end of each round r, each $C_i[0], C_i[1], \dots C_i[r]$ correctly represents the number of elements in buckets of hamming distance $0,1,\dots r$ from address i, for all addresses $i \in B$. We use proof by induction on the number of rounds, r.

Base Case: In round $r = 0$, the update rule $C_i[0] = b_i$ is clearly correct. For round $r = 1$, the update rule $C_i[1] = \sum_{j \in \mathcal{N}_i} C_j[0]$ is also clearly correct.

Inductive Hypothesis: Assume that at end of round $r-1$, each $C_i[0], C_i[1], \dots C_i[r-1]$ correctly represents the number of elements in buckets of hamming distance $0,1,\dots r-1$ from address i, for all addresses $i \in B$.

Algorithm 1 Aggregate-Counts

```
Require: Hash table with buckets $B$
 1: for round $r = 0, r++, r \leq t$ do
 2:   for hash address $i \in B$ do
 3:     if r == 0 then
 4:       $C_i[r] = b_i$
 5:     else if r ==1 then
 6:       $C_i[r] = \sum_{j \in \mathcal{N}_i} C_j[r-1]$
 7:     else
 8:       $C_i[r] = \frac{\sum_{j \in \mathcal{N}_i} C_j[r-1] - (t-r+2)C_i[r-2]}{r}$
 9:     end if
10:   end for
11: end for
12: Return $C_i$ for all $i \in B$
```

Inductive Step: We want to show that at round r, $C_i[r] = \frac{\sum_{j \in \mathcal{N}_i} C_j[r-1] - (t-r+2)C_i[r-2]}{r}$ correctly counts the number of elements in buckets at hamming distance r from bucket i.

Without loss of generality, we focus on hash address $\mathbf{0}_t$, the address with t (all) 0's. The hash addresses that are hamming distance r from $\mathbf{0}_t$ contain exactly r 1's.

We first take $\sum_{j \in \mathcal{N}_i} C_j[r-1]$, the sum of the elements that are hamming distance $r-1$ from $\mathbf{0}_t$'s immediate neighbors. However, this sum also includes elements that are $r-1$ distance from $\mathbf{0}_t$'s neighbors via $\mathbf{0}_t$ as an intermediate hop, which consequently are not distance r from $\mathbf{0}_t$, but rather $r-2$ distance from $\mathbf{0}_t$, since each neighbor is distance 1 from $\mathbf{0}_t$. We claim that there are exactly $(t-r+2)C_{\mathbf{0}_t}[r-2]$ of these elements that were included in the sum. Fix any hash address h that is distance $r-2$ from $\mathbf{0}_t$. Clearly h has exactly $(r-2)$ 1's in its address. There are $(t-r+2)$ 0's in h's address. Any neighbor j of $\mathbf{0}_t$ that has a 1 in any of those $t-r+2$ slots will report h as part of its count $C_j[r-1]$. There are $t-r+2$ such neighbors, therefore, we must adjust $\sum_{j \in \mathcal{N}_i} C_j[r-1]$ by subtracting $(t-r+2)C_i[r-2]$.

There is one other source of double-counting, which is that many neighbors j of $\mathbf{0}_t$ will include the same bucket of hamming distance r away from $\mathbf{0}_t$ as part of their $C_j[r-1]$ count. This over-counting can be quantified in the following way. Fix any hash address d that is distance r from $\mathbf{0}_t$. Clearly the sequence d contains exactly (r) 1's. So any neighbor j of $\mathbf{0}_t$ that contains a 1 in any one of those r slots will include d as part of its $C_j[r-1]$ count, and there are r such neighbors. So our final expression for $C_{\mathbf{0}_t}(r)$ is

$$C_{\mathbf{0}_t}[r] = \frac{\sum_{j \in \mathcal{N}_i} C_j[r-1] - (t-r+2)C_{\mathbf{0}_t}[r-2]}{r}$$

This same argument generalizes to any bucket i, so we conclude that our general update rule $C_i[r] = \frac{\sum_{j \in \mathcal{N}_i} C_j[r-1] - (t-r+2)C_i[r-2]}{r}$ is correct.

`Aggregate-Counts` proceeds in $t+1$ rounds in the outer loop. Each round iterates over 2^t buckets in the inner loop. An update for each bucket i looks at the neighbors $\mathcal{N}_i$ of i, and each hash address i has exactly t neighbors (each corresponding to one bit flip in the length t hash address). So `Aggregate-Counts` terminates in time $O(t^2 2^t)$.

When we want to compute the counts vectors for many tables, we invoke `Aggregate-Counts` for each of K tables so the overall runtime is $O(Kt^2 2^t)$.

B. Constructing a Uniform Sampler for Fixed Hamming Distances across Multiple Tables

We construct a sampler that, given a different hash address i^k for each table $k \in [K]$, and a certain set of hamming distances $\mathcal{I}$, returns a data point uniformly at random from the union of the set of elements in each table that are contained in buckets of hamming distance $\mathcal{I}$ from hash address i^k.

The high level implementation of our sampler works as follows. Given a set of hamming distances $\mathcal{I}$ that we are interested in and hash address i^k for each of our K tables, we know from Appendix A that we can construct C_i^k that gives the number of elements in buckets that are hamming distance $0, 1, \ldots t$ away from address i^k for each table k. Then for each table, we can add the relevant indices to obtain the total count for elements at hamming distances $\mathcal{I}$, that is, we can compute $\sum_{d \in \mathcal{I}} C_i^k[d]$ for each k. We choose to take a sample from table k^* with probability $\frac{\sum_{d \in \mathcal{I}} C_i^{k^*}[d]}{\sum_k \sum_{d \in \mathcal{I}} C_i^k[d]}$.

Now that we have fixed our choice of table k^*, we want to pick a particular hamming distance within $\mathcal{I}$ to sample from. This can be done using the counts vector for that table, in particular we choose the hamming distance $d^* \in \mathcal{I}$ with probability $\frac{C_i^{k^*}[d^*]}{\sum_{d \in \mathcal{I}} C_i^{k^*}[d]}$.

Having now fixed a table k^* and a particular hamming distance d^*, we introduce an algorithm `Hamming-Distance-Sampler` that generates a sample uniformly from the set of elements hashed to buckets at hamming distance d^* from address $\mathbf{0}_t$ (without loss of generality) in table k^*. Our algorithm uses the counts matrix M_i^k as the underlying data structure. Our main results says:

Theorem 3.2 (Sampler)**.** Given a set of K hash tables, each with 2^t hash buckets with addresses in $\{0,1\}^t$, a sampling scheme consisting of a data structure and a sampling algorithm can generate a sample uniformly at random from any fixed hash table k, an element at hamming distance d to hash address i. The data structure is a counts matrix that can be precomputed in preprocessing time $O(Kt^3 2^t)$, and the sampling algorithm `Hamming-Distance-Sampler` (Algorithm 2) generates a sample in time $O(t)$.

We first describe the implementation of the sampler in Section $B.1$ and then later describe the implementation for constructing the Counts Matrix M_i^k in Section $B.2$. Our main result follows from Lemma $B.1$ and Lemma $B.2$

B.1. Uniform Sampler for One Table

In this section, we describe our algorithm, `Hamming-Distance-Sampler` (Algorithm 2), which helps to generate a sample uniformly at random, from one hash table, an element from hamming distance d to hash address i. Suppose for each hash address i in the table we are given its counts matrix $M_i \in \mathbb{Z}^{(t+1)\times(t+1)}$ such that $M_i[s, a]$ gives a count of the number of elements hashed to buckets whose addresses are at hamming distance s away from i and share the same first a bits as address i. `Hamming-Distance-Sampler` uses this counts matrix to help decide which hash bucket to sample from.

`Hamming-Distance-Sampler` (Algorithm 2) chooses a target hash address that is hamming distance d from $\mathbf{0}_t$ by iteratively generating a bit pattern to XOR with the query hash address. Without loss of generality, suppose the query hash address is $\mathbf{0}_t$. We start from left to right. We set the first bit of the XOR mask to 1 with probability proportional to the number of elements at hamming distance d to $\mathbf{0}_t$ that have 1 as their first bit.

Now that we have decided on the first bit of the XOR mask, we move on to the second bit. Conditioned on our choice for the first bit, we make our second choice. If we had chosen 1 for the first bit of the mask, now we choose to set the second bit of the XOR mask with probability proportional to the number of elements at hamming distance d to $\mathbf{0}_t$ that have 11 as their first two bits, and we choose to set the second bit to 0 with probability proportional to the number of elements at hamming distance d to $\mathbf{0}_t$ that have 10 as their first two bits, and so on and so forth.

We continue until we arrive at a target hash address that is exactly hamming distance d from i, which is the output of `Hamming-Distance-Sampler`. After we choose our target sampling hash address, we sample uniformly at random the elements within that hash bucket. `Hamming-Distance-Sampler` is formally written as Algorithm 2.

B.1.1. Analysis of Algorithm

We prove the following guarantee for `Hamming-Distance-Sampler` (Algorithm 2):

Lemma B.1. *Suppose there are a total of D elements that are contained in buckets of hamming distance d from hash address $i \in \{0,1\}^t$, and bucket b which is hamming distance d from address i contains m elements. Then* `Hamming-Distance-Sampler` *(Algorithm 2) returns address b with probability $\frac{m}{D}$ in time $O(t)$.*

Once we have the output of `Hamming-Distance-Sampler`, which is a hash address b that was generated with probability $\frac{m}{D}$, then we can pick an element uniformly at random from within bucket b to generate a sample with uniform probability $\frac{1}{D}$.

Algorithm 2 Hamming Distance Sampler

```
Require: Hash table with buckets $B$, hash address $i$, hamming distance $d$, counts matrix $M_b$ for all buckets $b \in B$
 1: MASK $= \mathbf{0}_t$
 2: $g = 0$
 3: for round $r = 0, r$++$, r < t$ do
 4:     while $g < d$ do
 5:         $i' = i \oplus$ MASK
 6:         $i'' = i \oplus$ MASK $\oplus \mathbf{0}_r\mathbf{1}\mathbf{0}_{t-r-1}$
 7:         $p_r = \frac{M_{i''}[d-g-1,r+1]}{M_{i''}[d-g-1,r+1]+M_{i'}[d-g,r+1]}$
 8:         Flip a biased coin with probability $p_r$ of coming up heads. Let $f = 1$ if heads, $f = 0$ else.
 9:         if $f = 1$ then
10:             $g \leftarrow g + 1$          //Update the count of 1's already chosen
11:             MASK $\leftarrow$ MASK $\oplus \mathbf{0}_r\mathbf{1}\mathbf{0}_{t-r-1}$          //Update the MASK to make the $(r+1)$-th bit 1
12:         end if
13:     end while
14: end for
15: Return $i \oplus$ MASK
```

Proof. Without loss of generality, we assume that the hash address of interest $i = \mathbf{0}_t$.

We first notice that the set of all possible realizations of this algorithm can be represented as a binary tree with depth at most t, and each round can be viewed as traversing the binary tree. We first describe this tree. The root of the tree has label $\mathbf{0}_t$, and its value is the total number of elements in buckets at hamming distance d from $\mathbf{0}_t$. Its left child node has label $\mathbf{0}_t$ and its value is the total number of elements across all buckets at hamming distance d away from $\mathbf{0}_t$ that share the first bit (0). The root's right child node has label $\mathbf{1}\mathbf{0}_{t-1}$ and its value is the total number of elements across all buckets that are hamming distance $d-1$ away from $\mathbf{1}\mathbf{0}_{t-1}$.

In general, each node V at depth r can be expressed as a label, value pair (l, v), where the label l is a hash address, and if we let $g = d(i, l)$, and the value v is a count of elements at hamming distance $d-g$ away from the label hash address l. Its left child is labeled by l and its value is the total number of elements across all buckets at hamming distance $d-g$ away from the label of the node (l) that match the first $r+1$ bits as its label. The parent's right child is labeled by $l \oplus \mathbf{0}_r\mathbf{1}\mathbf{0}_{t-r-1}$. and its value is the total number of elements across all buckets that are hamming distance $d-g-1$ away from its label and that share the first $r+1$ bits as its label.

Clearly the leaves of the tree are the set of labels (bucket addresses) that are of hamming distance d away from i, and have values that correspond to the number of elements in each bucket. We also note that the label of each node corresponds to the XOR mask in our algorithm, and the value corresponds to an entry in the M_i^k.

We now analyze the correctness of `Hamming-Distance-Sampler` using proof by induction. We start at the root of the tree, with label $\mathbf{0}_t$. We want to show that the probabilities that the XOR mask takes on a specific value at the end of round r for $r = 0, \ldots t$ lead to uniform probabilities of choosing an element hashed to a bucket at hamming distance d from $\mathbf{0}_t$.

Suppose further there are a total of D elements that are contained in buckets of hamming distance d from hash address $\mathbf{0}_t$.

Base Case: After round $r = 0$, the probability that the XOR mask becomes $\mathbf{1}\mathbf{0}_{t-1}$ is $\frac{M_{\mathbf{1}\mathbf{0}_{t-1}}[d-1,1]}{M_{\mathbf{0}_t}[d,1]+M_{\mathbf{1}\mathbf{0}_{t-1}}[d-1,1]} = \frac{M_{\mathbf{1}\mathbf{0}_{t-1}}[d-1,1]}{D}$.

Inductive Hypothesis: Assume that at end of round $r-1$, the probability that we reach a certain node at level r with label (XOR mask) l, and with g_l 1's in the mask, is $\frac{M_l[d-g_l,r]}{D}$.

Inductive Step: We want to show that at the end of round r, the probability that we reach a certain node at level $r+1$ with label c (for child), and with g_c 1's in the mask, is $\frac{M_c[d-g_c,r+1]}{D}$.

This follows directly from the inductive hypothesis. Note that at the end of round $r-1$, we have reached a certain node at level r with label (XOR mask) l, and with g_l 1's in the mask, is $\frac{M_l[d-g_l,r]}{D}$. From this node, the probability of reaching the left child lc is $\frac{M_{lc}[d-g_{lc},r+1]}{M_l[d-g_l,r]}$ and the probability of reaching the right child is $\frac{M_{rc}[d-g_{rc},r+1]}{M_l[d-g_l,r]}$. Multiplying this by the probability that we reach the parent from the inductive hypothesis gives the proof of the inductive step.

Since there are a total of D elements that are contained in buckets of hamming distance d from hash address $\mathbf{0}_t$, and bucket b, which is hamming distance d from address i, contains m elements. The probability of reaching b by traversing down this tree is $\frac{m}{D}$.

Additionally, this algorithm takes $O(t)$ time to produce the hash address b. This is clear since the tree has depth at most t, and each step in the traversal is a constant time operation. □

B.2. Computing the Counts Matrix

Since `Hamming-Distance-Sampler` (Algorithm 2) requires a matrix $M_i \in \mathbb{Z}^{(t+1)\times(t+1)}$ such that $M_i[s,a]$ gives a count of the number of elements that are hamming distance s away from i and share the same first a bits as address i, we show how to precompute such a counts matrix in time $O(Kt^3 2^t)$ for each $i \in \{0,1\}^t$ over the K hash tables.

Lemma B.2. *The counts matrix $M_i^k \in \mathbb{Z}^{(t+1)\times(t+1)}$ can be computed in time $O(Kt^3 2^t)$ for each $i \in \{0,1\}^t$ and $k \in \{1, \dots K\}$.*

Proof. We use the algorithm that we develop in Appendix A, `Aggregate-Counts` (Algorithm 1), to compute matrix M_i.

Now, to use `Aggregate-Counts`, we observe that $M_i[s,a]$ for all $s \in \{0,1,\dots t\}$ is just another instance of the `Aggregate-Counts` problem, restricted to the case where we only consider buckets that match i on the first a bits. Clearly, by the Appendix A Main Theorem we can compute $M_i[s][0]$ for all $s \in \{0,\dots t\}$ (the entire column) simultaneously for all i in time $O(t^2 2^t)$.

In fact, fixing each $a \in \{0,1,\dots t\}$, it is possible to compute $M_i[s][a]$ for all $s \in \{0,\dots t\}$ (the entire column) simultaneously for all i in time $O(t^2 2^t)$. This is because in our updates for each i, we can just consider the buckets that match i on the first a bits. We can invoke `Aggregate-Counts` using neighbor buckets $\mathcal{N}_i(a)$, and the number of rounds would be $t-a$. The runtime to update each column of M_i (fixing a and over all i) is $O(t^2 2^t)$ (one invocation of `Aggregate-Counts`), so the total runtime to compute M_i^k over all i is $O(t^3 2^t)$. Repeating for each of K tables yields the final runtime of $O(Kt^3 2^t)$. □

C. Omitted Proofs From *Local Density Estimation in High Dimensions*

C.1. Preprocessing

Lemma 3.1 (Expectation of Z)**.** *The expectation of Z over the random choice of hash functions is $|\mathcal{A}_q|$, i.e. $\mathbb{E}(Z) = |\mathcal{A}_q|$. The expectation of Z given a specific realization of hash functions, or equivalently, given W, is $\mathbb{E}(Z|W) = W$.*

Proof. We sample each $x \in \mathcal{A}_q$ with probability $\frac{\sum_{k=1}^K \mathbb{1}(x\in\mathcal{B}_q^k(\mathcal{I}))}{\sum_{k=1}^K C_q^k(\mathcal{I})}$. Given $W = \sum_{x\in\mathcal{A}_q} \frac{\sum_{k=1}^K \mathbb{1}(x\in\mathcal{B}_q^k(\mathcal{I}))}{K\cdot p(x)}$, we have:

$$\begin{aligned}
\mathbb{E}(Z|W) &= \sum_{x\in\mathcal{A}_q} \frac{\sum_{k=1}^K C_q^k(\mathcal{I})}{K\cdot p(x)} \cdot \frac{\sum_{k=1}^K \mathbb{1}(x\in\mathcal{B}_q^k(\mathcal{I}))}{\sum_{k=1}^K C_q^k(\mathcal{I})}\Bigg| W \\
&= \sum_{x\in\mathcal{A}_q} \frac{\sum_{k=1}^K \mathbb{1}(x\in\mathcal{B}_q^k(\mathcal{I}))}{K\cdot p(x)}\Bigg| W \\
&= W
\end{aligned}$$

Now,

$$\begin{aligned}\mathbb{E}(W) &= \frac{1}{K}\sum_{k=1}^{K}\sum_{x\in\mathcal{A}_q}\frac{\mathbb{E}(\mathbb{1}(x\in\mathcal{B}_q^k(\mathcal{I})))}{p(x)}\\ &= \frac{1}{K}\sum_{k=1}^{K}\sum_{x\in\mathcal{A}_q}\frac{p(x)}{p(x)} = |\mathcal{A}_q|\end{aligned}$$

Then clearly, $\mathbb{E}(Z|W) = W$ and $\mathbb{E}(Z) = \mathbb{E}(\mathbb{E}(Z|W)) = |\mathcal{A}_q|$. □

Lemma 3.2 (Variance of W)**.** $\sigma^2(W) = \frac{1}{K}\sum\limits_{x,y\in\mathcal{A}_q}\left(\frac{p(x,y)}{p(x)p(y)} - 1\right)$

Proof. We want to compute:

$$\begin{aligned}\mathbb{E}[W^2] = \frac{1}{K^2}\sum_{k=1}^{K}\sum_{\substack{x\in\mathcal{A}_q\\ y\in\mathcal{A}_q}}\frac{\mathbb{E}(\mathbb{1}(x\in\mathcal{B}_q^k(\mathcal{I}), y\in\mathcal{B}_q^k(\mathcal{I})))}{p(x)p(y)}+\\ \frac{1}{K^2}\sum_{k=1}^{K}\sum_{l\neq k}\sum_{\substack{x\in\mathcal{A}_q\\ y\in\mathcal{A}_q}}\frac{\mathbb{E}[\mathbb{1}(x\in\mathcal{B}_q^k(\mathcal{I}))\mathbb{1}(y\in\mathcal{B}_q^l(\mathcal{I}))]}{p(x)p(y)}\\ = \frac{1}{K}\sum_{x,y\in\mathcal{A}_q}\frac{p(x,y)}{p(x)p(y)} + \frac{1}{K^2}\sum_{k=1}^{K}\sum_{l\neq k}\sum_{\substack{x\in\mathcal{A}_q\\ y\in\mathcal{A}_q}}\frac{p(x)p(y)}{p(x)p(y)}\\ = \frac{1}{K}\sum_{x,y\in\mathcal{A}_q}\frac{p(x,y)}{p(x)p(y)} + \left(1-\frac{1}{K}\right)|\mathcal{A}_q|^2\end{aligned}$$

Since $\sigma^2(W) = \mathbb{E}[W^2] - (\mathbb{E}[W])^2$, we appeal to Lemma 3.1 to conclude:

$$\sigma^2(W) = \frac{1}{K}\sum_{x,y\in\mathcal{A}_q}\left(\frac{p(x,y)}{p(x)p(y)} - 1\right).$$

□

C.2. Sampling

Lemma 4.1 (Variance of Estimator)**.**

$$\mathbb{E}\left[\left(\frac{\sum_{i=1}^{S} Z_i}{S} - |\mathcal{A}_q|\right)^2\right] \leq \frac{\mathbb{E}[Z^2]}{S} + \sigma^2(W)$$

Proof. The variance can be expressed as:

$$\begin{aligned}
\mathbb{E}\left[\left(\frac{\sum_{i=1}^{S} Z_i}{S} - |\mathcal{A}_q|\right)^2\right] &= \mathbb{E}\left[\left(\frac{\sum_{i=1}^{S}(Z_i - W)}{S} + (W - |\mathcal{A}_q|)\right)^2\right] \\
&= \mathbb{E}\left[\frac{\sum_{i=1}^{S}(Z_i - W)^2}{S^2} + (W - |\mathcal{A}_q|)^2\right] \\
&= \mathbb{E}\left[\frac{(Z - W)^2}{S} + (W - |\mathcal{A}_q|)^2\right] \\
&= \frac{\mathbb{E}[Z^2] - \mathbb{E}[W^2]}{S} + \mathbb{E}[W^2] - |\mathcal{A}_q|^2 \\
&\leq \frac{\mathbb{E}[Z^2]}{S} + \mathbb{E}[W^2] - |\mathcal{A}_q|^2 \\
&= \frac{\mathbb{E}[Z^2]}{S} + \sigma^2(W)
\end{aligned}$$

□

Lemma 4.2 (Variance of Z)**.**

$$\mathbb{E}[Z^2] = \sum_{x \in \mathcal{A}_q} \sum_{y \in \mathcal{D}} \left[\frac{p(x,y)}{K \cdot p(x)^2} + \left(1 - \frac{1}{K}\right)\frac{p(y)}{p(x)}\right]$$

Proof. To analyze the second moment of Z, as with our first moment analysis of Z, we first condition on fixing the hash tables, so given $g_1, \dots g_K$, we know which elements of interest in $\mathcal{A}_q$ end up in our hamming distance set of interest $\mathcal{I}$.

$$\begin{aligned}
\mathbb{E}[Z^2|g_1, \dots g_K] &= \frac{1}{K^2}\left(\sum_{k=1}^{K} C_q^k(\mathcal{I})\right)\left(\sum_{k=1}^{K} \sum_{x \in \mathcal{A}_q \cap \mathcal{B}_q^k(\mathcal{I})} \frac{1}{p(x)^2}\right) \\
&= \frac{1}{K^2}\left[\sum_{k=1}^{K} \sum_{\substack{x \in \mathcal{A}_q \cap \mathcal{B}_q^k \\ y \in \mathcal{B}_q^k}} \frac{1}{p(x)^2} + \sum_{k=1}^{K}\sum_{l \neq k} \sum_{\substack{x \in \mathcal{A}_q \cap \mathcal{B}_q^k \\ y \in \mathcal{B}_q^l}} \frac{1}{p(x)^2}\right]
\end{aligned}$$

Now using the fact that $\mathbb{E}[Z^2] = \mathbb{E}[\mathbb{E}[Z^2|g_1, \dots g_K]]$, we have:

$$\begin{aligned}
\mathbb{E}[Z^2] &= \frac{1}{K^2}\,\mathbb{E}\left[\sum_{k=1}^{K} \sum_{\substack{x \in \mathcal{A}_q \\ y \in \mathcal{D}}} \frac{\mathbb{1}(x \in \mathcal{B}_q^k(\mathcal{I}))\mathbb{1}(y \in \mathcal{B}_q^k(\mathcal{I}))}{p(x)^2}\right] \\
&+ \frac{1}{K^2}\,\mathbb{E}\left[\sum_{k=1}^{K}\sum_{l \neq k} \sum_{\substack{x \in \mathcal{A}_q \\ y \in \mathcal{D}}} \frac{\mathbb{1}(x \in \mathcal{B}_q^k(\mathcal{I}))\mathbb{1}(y \in \mathcal{B}_q^l(\mathcal{I}))}{p(x)^2}\right] \\
&= \frac{1}{K}\sum_{\substack{x \in \mathcal{A}_q \\ y \in \mathcal{D}}} \frac{p(x,y)}{p(x)^2} + \left(1 - \frac{1}{K}\right)\sum_{\substack{x \in \mathcal{A}_q \\ y \in \mathcal{D}}} \frac{p(x)p(y)}{p(x)^2} \\
&= \sum_{x \in \mathcal{A}_q} \sum_{y \in \mathcal{D}} \left[\frac{p(x,y)}{K \cdot p(x)^2} + \left(1 - \frac{1}{K}\right)\frac{p(y)}{p(x)}\right] \qquad (3)
\end{aligned}$$

□