1:

(a) I use $k = 12$ For the hashing function:

$$p = (a \cdot h + b) \mod d$$

h is the original value by convert string to integer, for the first set of the hashing:

$$a = 775169054918279404$$

$$b = 1758426461858698312$$

$$d = 2305843009213693951$$

Signature vector of r1:

$$\begin{bmatrix} 4005303368 \\ 4290846341 \\ 1703301249 \\ 2876537340 \\ 3634877716 \\ 2252917204 \\ 986026652 \\ 3925436996 \\ 1849836273 \\ 3343156310 \\ 3447817223 \\ 2225448249 \end{bmatrix}$$

Signature vector of r2:

$$\begin{bmatrix} 3808290889 \\ 4183569548 \\ 3517292419 \\ 3065164290 \\ 3574749975 \\ 3625436121 \\ 3536807325 \\ 4046572363 \\ 1380444918 \\ 3248330072 \\ 3287169547 \\ 3898623806 \end{bmatrix}$$

Signature vector of r3:

$$\begin{bmatrix} 98973768 \\ 441755787 \\ 245669764 \\ 478253569 \\ 779415317 \\ 198283734 \\ 59674525 \\ 48417611 \\ 77630194 \\ 621852252 \\ 1169392648 \\ 195184444 \end{bmatrix}$$

(b)

Hamming Distance:

Here I use hamming distance for two 1D vector, which is the proportion of disagreeing components

r1 and r2: 1.0

r1 and r3: 1.0

r2 and r3: 1.0

(c)

For LSH, I use byteswap function as my hashing function. The input of the hashing will be the $v1 = \begin{bmatrix} 98973768 \\ 441755787 \\ 245669764 \\ 478253569 \end{bmatrix}$ and $v2 = \begin{bmatrix} 779415317 \\ 198283734 \\ 59674525 \\ 48417611 \end{bmatrix}$.

I first convert vector as a binary array, converting little-endian values to big-endian (and vice versa) for each element. After this operation, the concatenated of the binary array will use as the hashing key. So we can assume that all the key are unique

(d) No, although the 2nd band of vector 1 and the 3rd band of vector 2 are same, but LSH only put the same hashing value in same band into same bucket, so these two vector will hashing into diffrent bucket in all 3 bands.

After apply hashing in LSH:

$$v1 = \begin{bmatrix} b'\backslash x00\backslash x00\backslash x00\backslash x00\backslash x00\backslash x00\backslash x00\backslash x0b\backslash x00\backslash x00\backslash x00\backslash x16\backslash x00\backslash x00\backslash x00\backslash x03' \\ b'\backslash x00\backslash x00\backslash x00\backslash x05\backslash x00\backslash x00\backslash x00\backslash x06\backslash x00\backslash x00\backslash x00\backslash t\backslash x00\backslash x00\backslash x00- ' \\ b'\backslash x00\backslash x00\backslash x00b\backslash x00\backslash x00\backslash x00\backslash x00\backslash x00\backslash x00\backslash x00\backslash x01\backslash x00\backslash x00\backslash x00\backslash x07' \end{bmatrix}$$

$$v2 = \begin{bmatrix} b'\backslash x00\backslash x00\backslash x00\backslash x0b\backslash x00\backslash x00\backslash x00\backslash t\backslash x00\backslash x00\backslash x00\backslash x03\backslash x00\backslash x00\backslash x00\backslash x04' \\ b'\backslash x00\backslash x00\backslash x00b\backslash x00\backslash x00\backslash x00\backslash x00\backslash x00\backslash x00\backslash x00\backslash x01\backslash x00\backslash x00\backslash x00\backslash x07' \\ b'\backslash x00\backslash x00\backslash x00\backslash x17\backslash x00\backslash x00\backslash x00\backslash x0f\backslash x00\backslash x00\backslash x00\backslash x00\backslash x00\backslash x00\backslash x00\backslash x1f' \end{bmatrix}$$

2:

(a)

Using same kmer size with different number of permutations

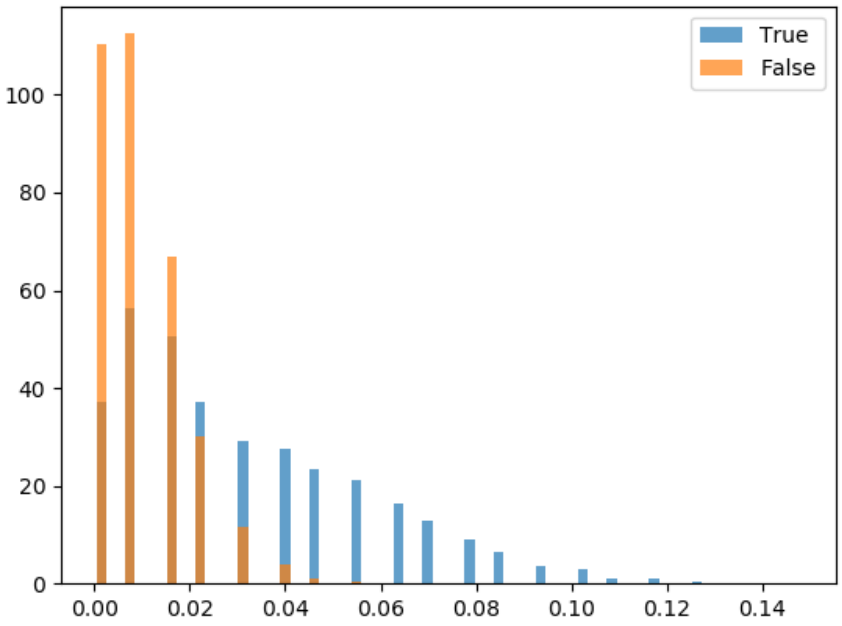

Figure 1: Identity for k =10 with 128 permutations

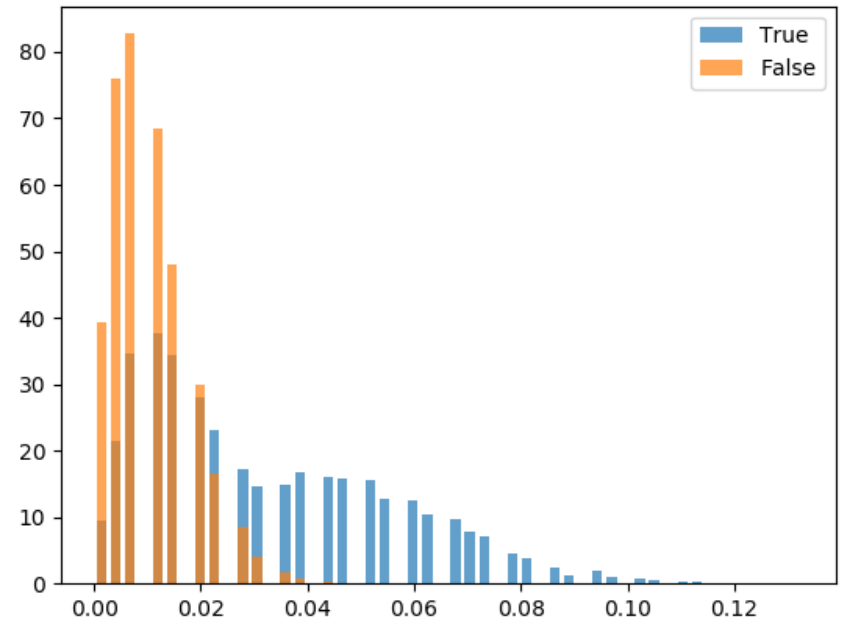

Figure 2: Identity for k =10 with 256 permutations

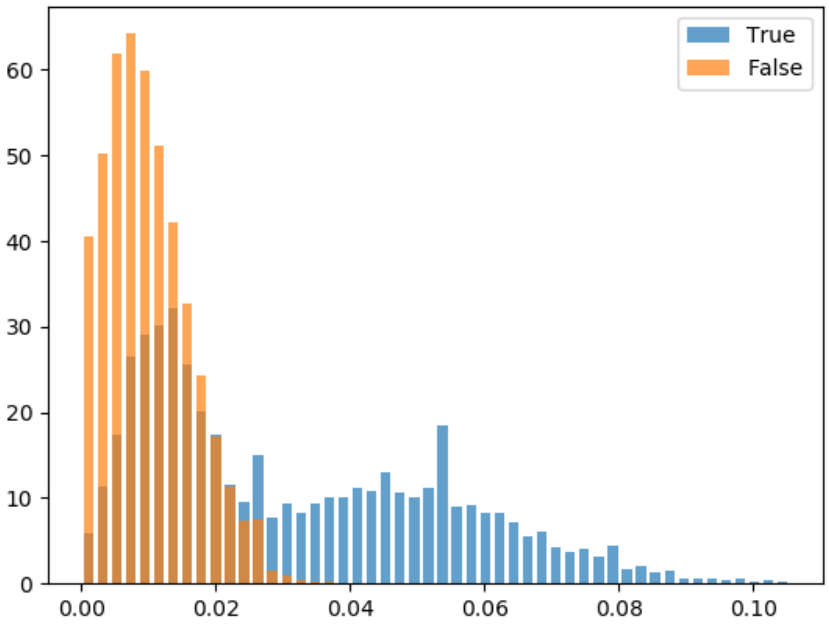

Figure 3: Identity for k =10 with 512 permutations

We can see with increasing number of permutations, we have higher resolution to distinguish the true overlap with other non-overlap pair.

The Jaccard similarities are showed below

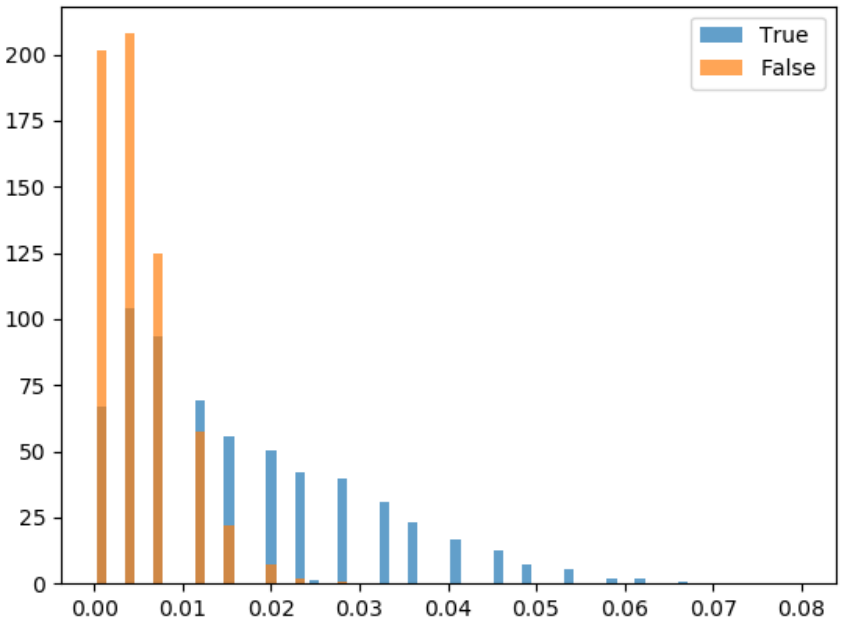

Figure 4: Jaccard similarity for k =10 with 128 permutations

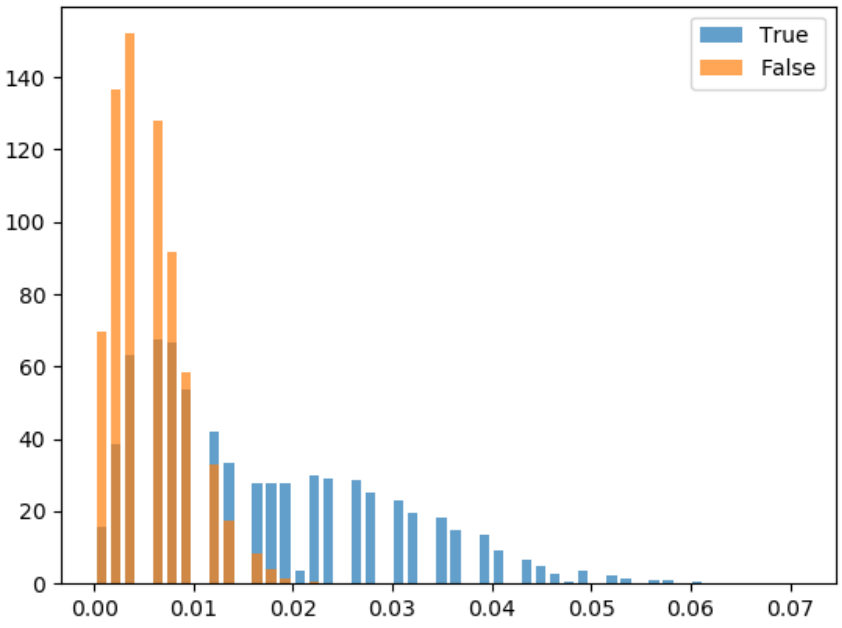

Figure 5: Jaccard similarity for k =10 with 256 permutations

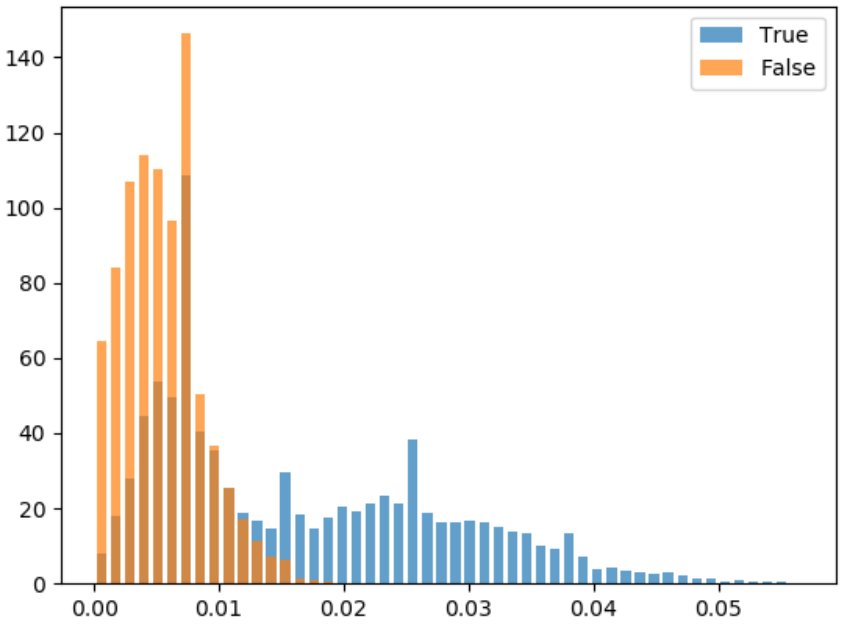

Figure 6: Jaccard similarity for k =10 with 512 permutations

From these figures, we can found that Jaccard similarity is lower than identities.

I also test the size k impact

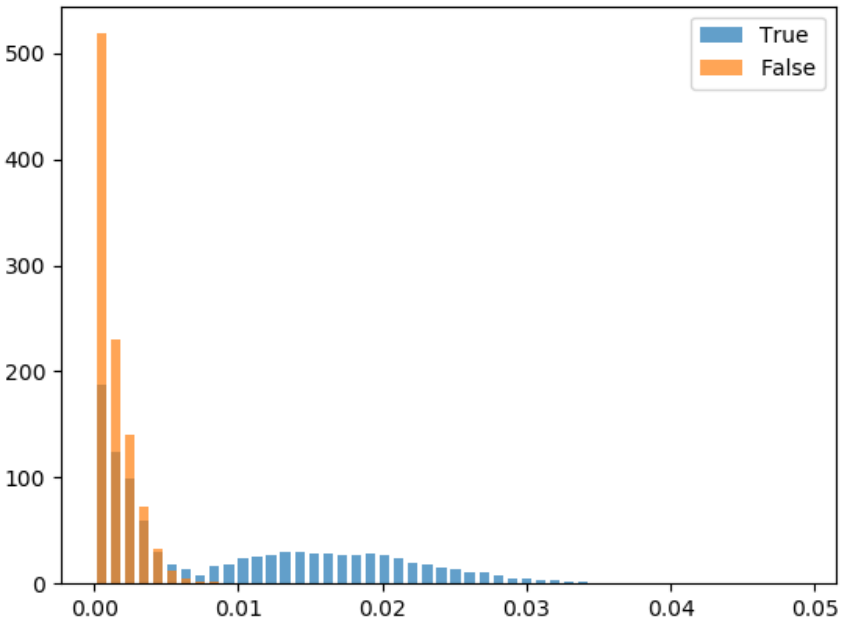

Figure 7: Identity for k =11 with 512 permutations

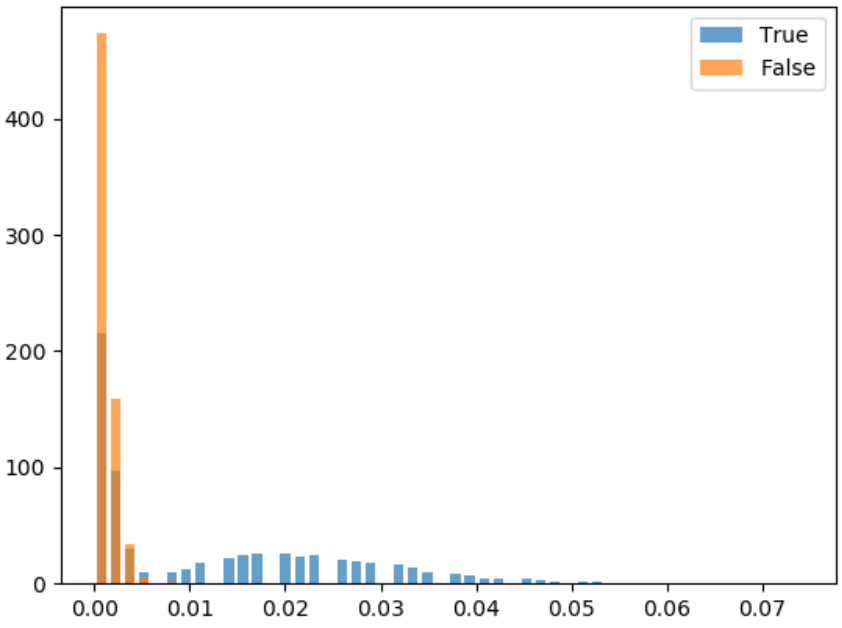

Figure 8: Identity for k =12 with 512 permutations

With large k, the threshold to distinguish the true and false become less.

I will choose original LSH which should be close to identity.

(b)

The false positive rate is given by:

$$FP = \int_0^s 1 - (1 - s^r)^b$$

The false negative rate is given by:

$$FN = \int_s^1 1 - (1 - (1 - s^r)^b)$$

I decide to use $k = 12$ and $s = 0.01$, the optimized $b = 166$ and$r = 1$ which minimize the sum of FP and FN. The estimated FN = 1.1239e-05 and estimated FP = 0

(c) I follow the identity and will compare under same band. LSH hashing function I used is byteswap function from numpy library. Following the optimized b and r, if we can hashing the size r vector from two different read under the same band to one bucket, we will report these two reads as one overlap pair.

(e) Using $k = 12$ with 512 permutations and threshold = 0.01, I got the following result:

sensitivity 0.32540324499214923

FPR 0.3302799701435846

accuracy 0.04372734189679094

F1 0.07709477054188414

Filtration Rate: 0.33006366980337704

The false positive rate obviously much larger than our theory result, I believe this is caused by high error rates of our DNA reads

(f)

Because we artificially construct 5000 overlap size threshold, so we can see that a lot of "false positive" are reposted because they just below 5000 threshold.

Noticed here I only count all the pairs that can found overlap sizes.

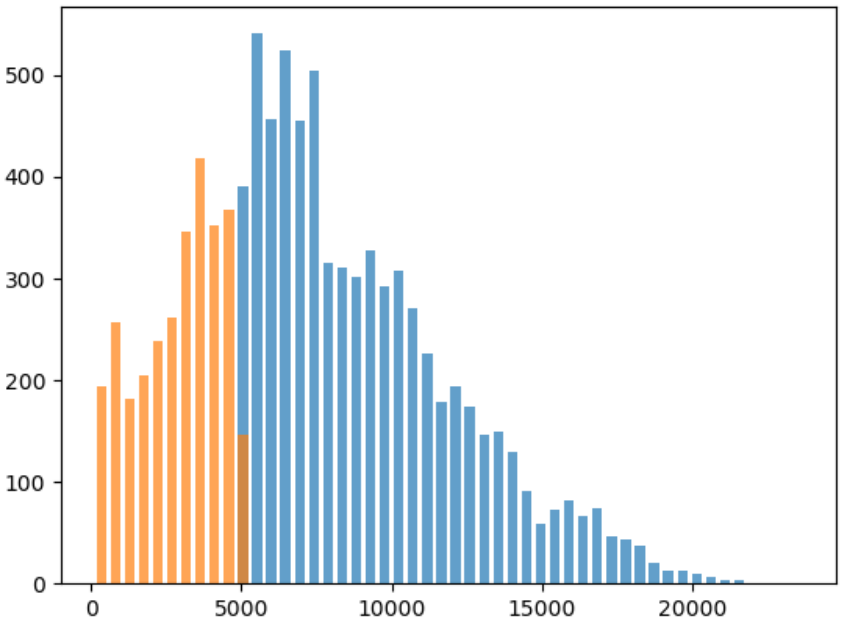

Figure 9: Overlap size comparison of true positive (blue) and false positive (orange)

3:

I use k = 10 with 1024 permutations and 0.04 threshold. Optimized $b = 512$ and $r = 2$, respectively

Total pairs: 233586.0, Expected Overlap: 22220, Total report: 35765, True Positive: 4094

sensitivity 0.18424842484248424

FPR 0.1498396146967819

accuracy 0.11446945337620579

F1 0.14120893334483056

Filtration Rate: 0.15311277216956495

4:

I plot the number of corrected clusters against size of that cluster.

With larger k, we can see we have more clusters and there are more and more correct clusters. And in general, the larger cluster tends to has error.

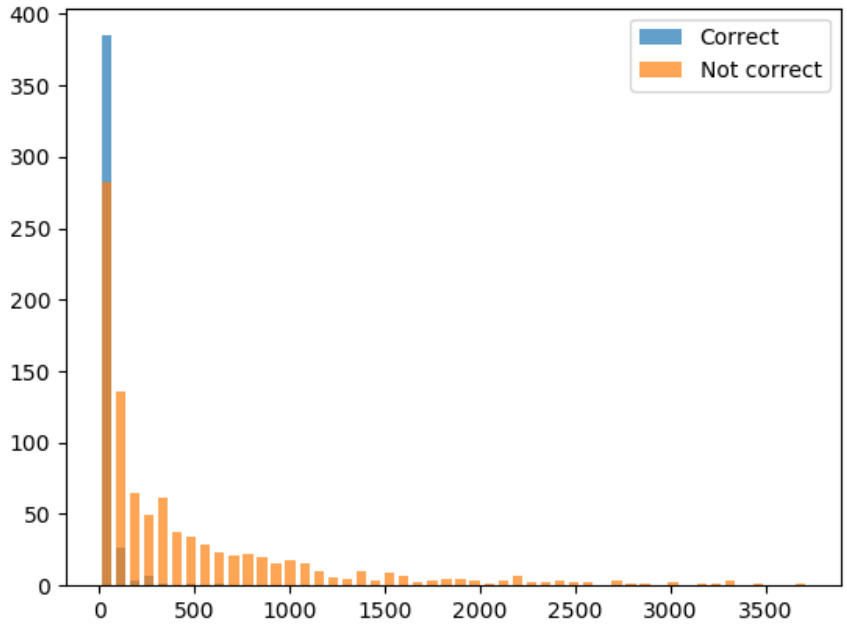

Figure 10: Number of clusters against cluster size with k = 6 and permutation = 32

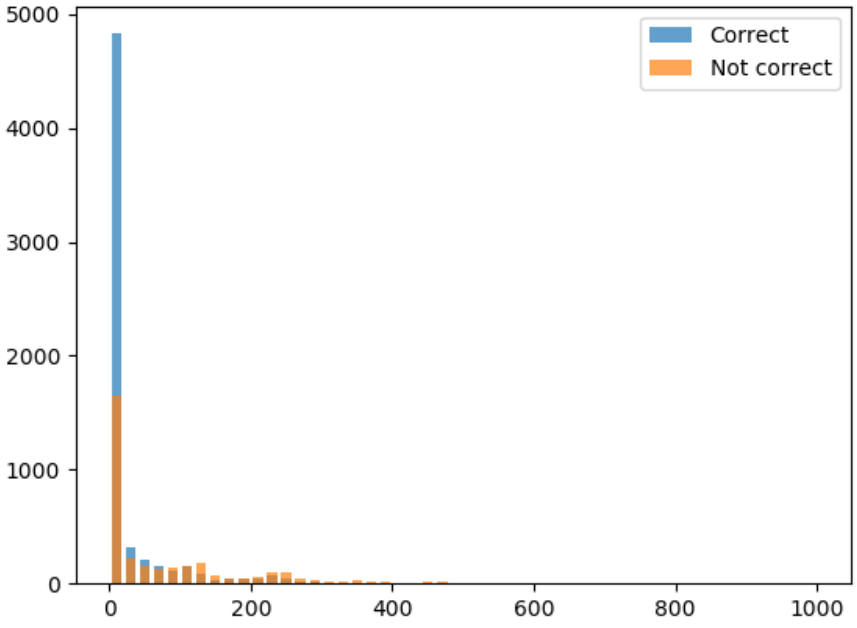

Figure 11: Number of clusters against cluster size with k = 8 and permutation = 32

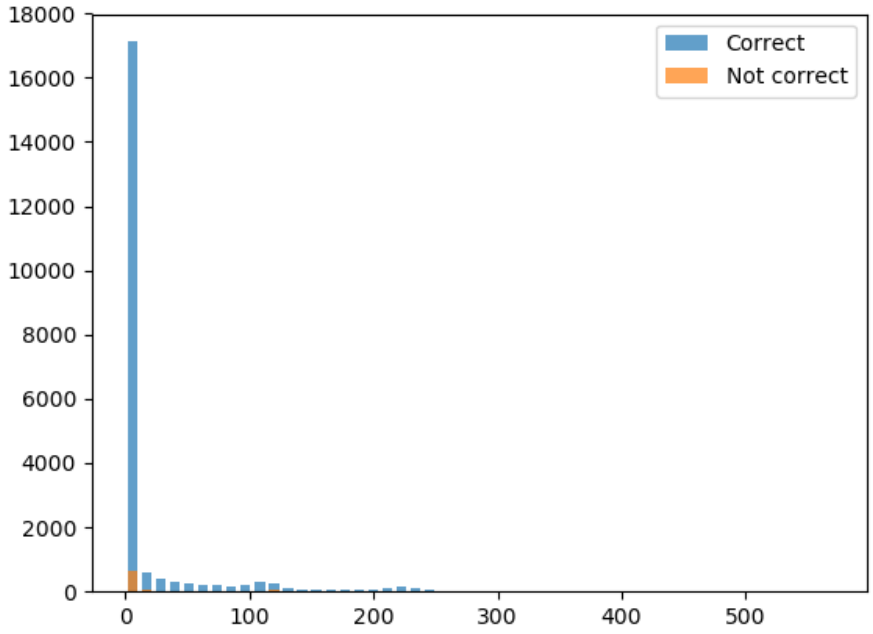

Figure 12: Number of clusters against cluster size with k = 10 and permutation = 32

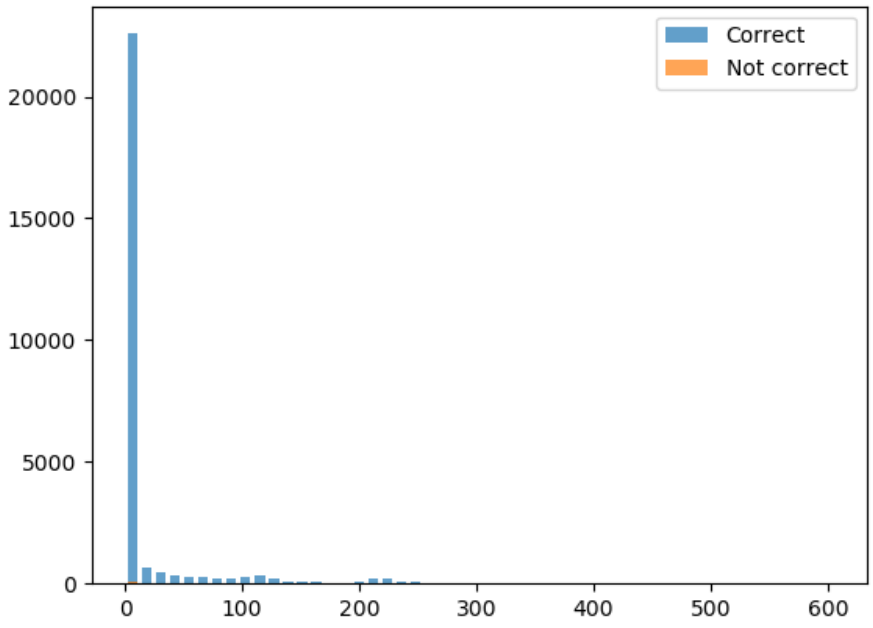

Figure 13: Number of clusters against cluster size with k = 12 and permutation = 32

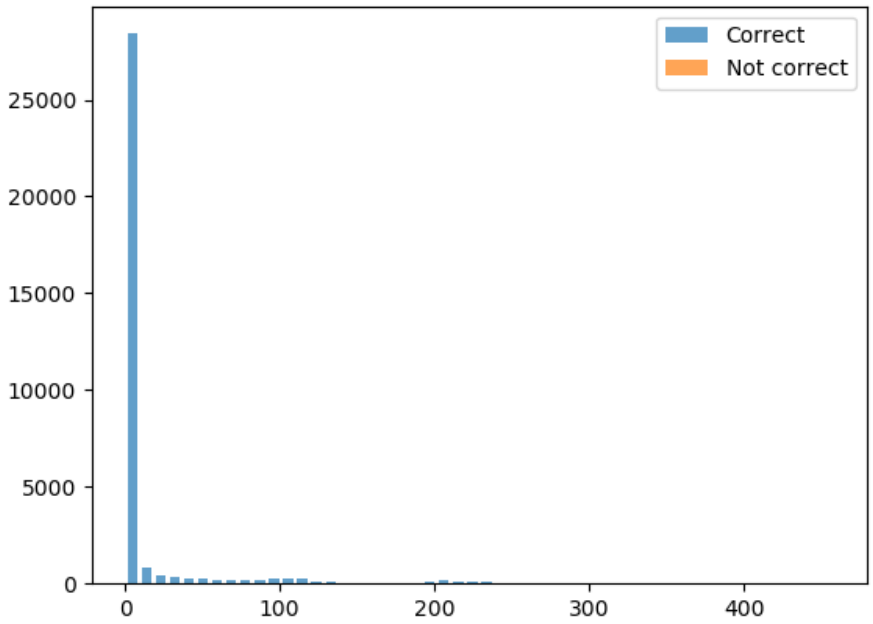

Figure 14: Number of clusters against cluster size with k = 16 and permutation = 32