# Space-efficient set differences with applications to Jaccard estimation

Yoshihiro Shibuya[1], Djamal Belazzougui[2] and Gregory Kucherov[1]

[1] LIGM, CNRS, Univ. Gustave Eiffel, Marne-la-Vallée, France
[2] CAPA, DTISI, Centre de Recherche sur l'Information Scientifique et Technique, Algiers, Algeria

# Context

Bottom-$s$ minHash sketches of size $s = 5$.

- $J_{AB} = \frac{|A \cap B|}{|A \cup B|} \approx \frac{|S(A \cup B) \cap S(A) \cap S(B)|}{|S(A \cup B)|}$

Problem:

- If $A \approx B$ choosing small $s$ leads to $J_{AB} = 1$
- Increasing $s$ nullies the space advatages of sketching

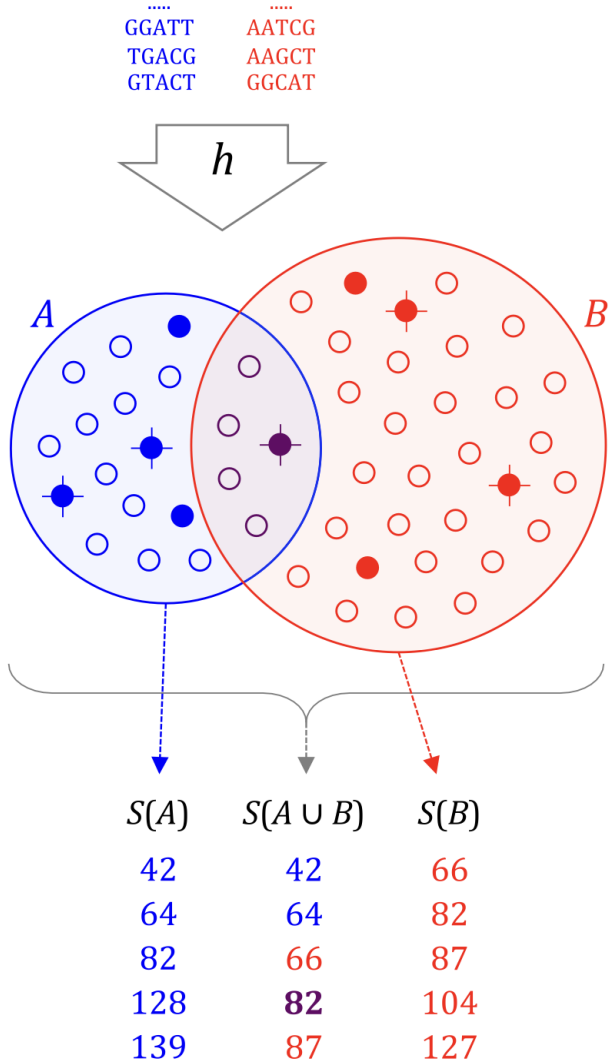



Figure: Image taken from [Ondov et al., 2016]

# This work

**Goal**:

- Estimate Jaccard similarity of very similar sequences using small space

**Assumptions:**

- All sequence pairs are of high similarity
- Storing all $k$-mer sets explicitly is not possible

**Ideas:**

- Use *Invertible Bloom Look-up Tables* (IBLTs) instead of minHash sketches
- Combine IBLTs with syncmers for further space reductions

# Invertible Bloom Look-up Tables

Invertible Bloom Look-up Tables (IBLTs) are inspired by Bloom filters:

- A table $T$ of $m$ buckets
- $r$ (pair-wise) independent hash functions $h_1(\cdot), \cdots, h_r(\cdot)$ mapping elements to $[1, \cdots, m]$
- An additional (global) hash function $h_e(\cdot)$

# Invertible Bloom Look-up Tables (cont.)

- Each bucket contains:
  - A counter $C$
  - A hash field $H$
  - A payload $P$

| C | H | P |
|---|---|---|
| 0 | 0 | 0 |
| ... | ... | ... |
| 0 | 0 | 0 |

# Invertible Bloom Look-up Tables (cont.)

- Each bucket contains:
  - A counter $C$
  - A hash field $H$
  - A payload $P$

(The utility of $H$ and $h_e(\cdot)$ will be explained later)

| C | H | P |
|---|---|---|
| 0 | 0 | 0 |
| ... | ... | ... |
| 0 | 0 | 0 |

# Invertible Bloom Look-up Tables (cont.)

- Each bucket contains:
  - A counter $C$
  - A hash field $H$
  - A payload $P$

(The utility of $H$ and $h_e(\cdot)$ will be explained later)

- insertions
- deletions

| C | H | P |
|---|---|---|
| 0 | 0 | 0 |
| ... | ... | ... |
| 0 | 0 | 0 |

# Invertible Bloom Look-up Tables (cont.)

- Each bucket contains:
  - A counter $C$
  - A hash field $H$
  - A payload $P$

(The utility of $H$ and $h_e(\cdot)$ will be explained later)

- insertions
- deletions
- **listing** of the elements they contain
  - Succeeds (with high probability, w.h.p.) depends on the number of stored elements compared to IBLT size
  - An over-charged IBLT can be listed again if enough elements are removed

| C | H | P |
|---|---|---|
| 0 | 0 | 0 |
| ... | ... | ... |
| 0 | 0 | 0 |

# IBLT: dimensioning

- IBLTs are closely related to $r$-hypergraphs
  - nodes are buckets
  - edges are the elements stored
- Listing is equivalent to peeling the hypergraph
  - a $r$-hypergraph of $m$ nodes and $n$ edges is peelable w.h.p. iff $m > c_r n$
  - $c_r$ is a peelability threshold (constant)
  - for $r = 3$, $c_r \approx 1.23$ [Goodrich and Mitzenmacher, 2011]

# IBLT: insertions and deletions

Insertions

**Data:** new element $p$, IBLT $T$
**foreach** $j$ *in* $[1, \cdots, r]$ **do**
  $i \leftarrow h_j(p)$;
  $T[i].C \leftarrow T[i].C + 1$;
  $T[i].H \leftarrow T[i].H \oplus h_e(p)$;
  $T[i].P \leftarrow T[i].P \oplus p$;
**end**

Deletions

**Data:** element $p$ to be deleted, IBLT $T$
**foreach** $j$ *in* $[1, \cdots, r]$ **do**
  $i \leftarrow h_j(p)$;
  $T[i].C \leftarrow T[i].C - 1$;
  $T[i].H \leftarrow T[i].H \oplus h_e(p)$;
  $T[i].P \leftarrow T[i].P \oplus p$;
**end**

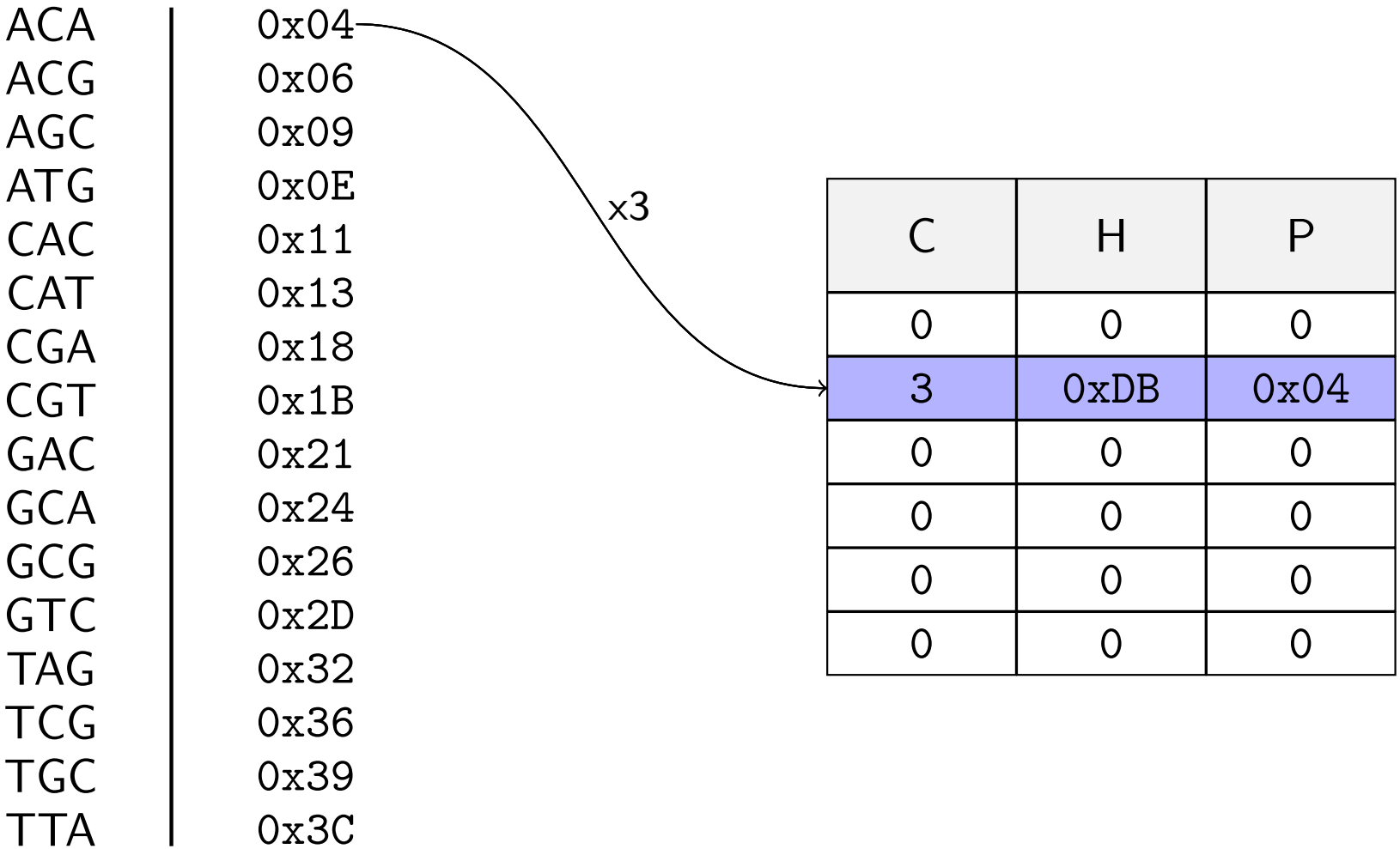
Since $\oplus$ (bitwise XOR) is the inverse of itself deleting elements from fields $H$ and $P$ is the same as inserting them.

# IBLT: construction

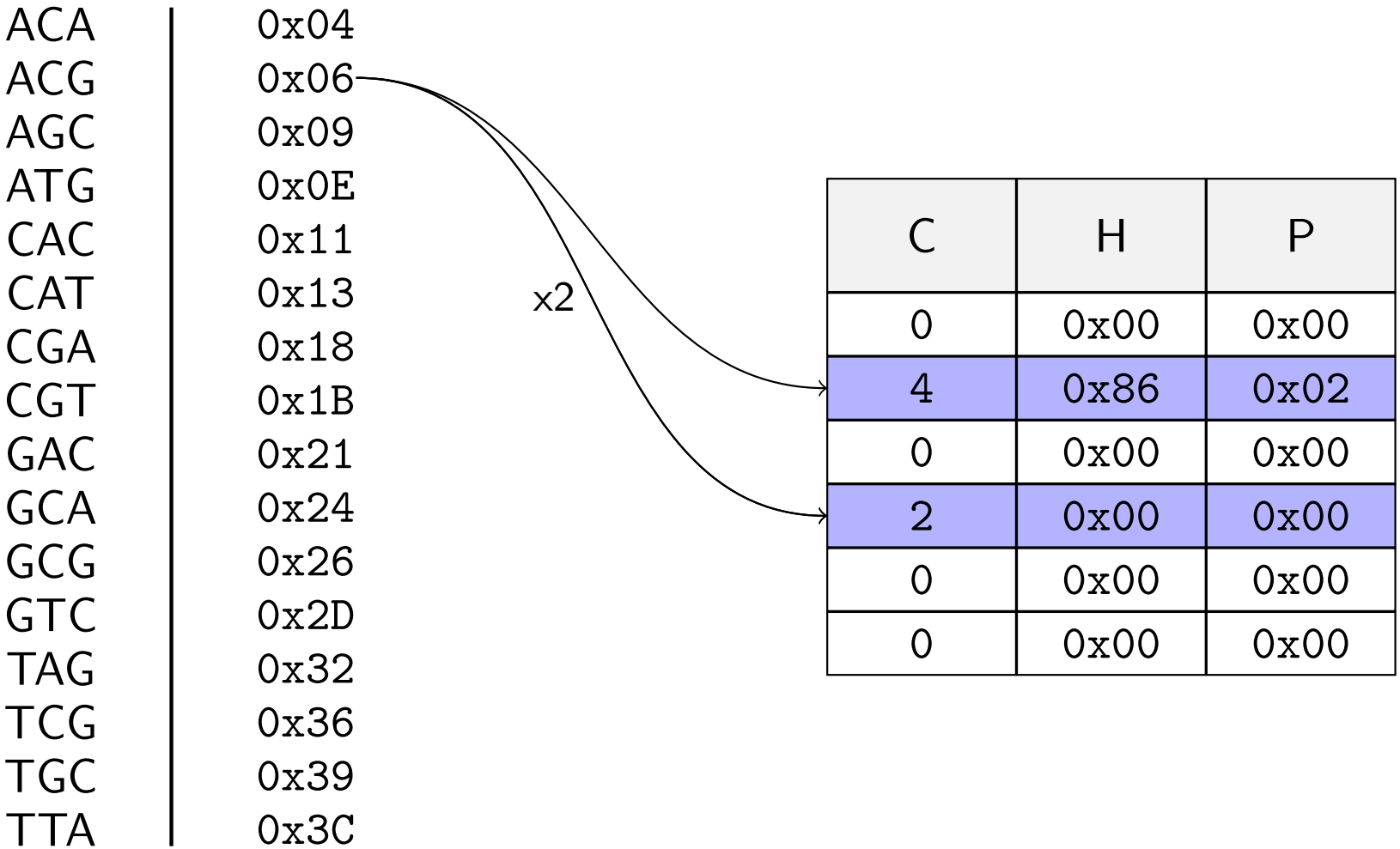
| | |
|---|---|
| ACA | 0x04 |
| ACG | 0x06 |
| AGC | 0x09 |
| ATG | 0x0E |
| CAC | 0x11 |
| CAT | 0x13 |
| CGA | 0x18 |
| CGT | 0x1B |
| GAC | 0x21 |
| GCA | 0x24 |
| GCG | 0x26 |
| GTC | 0x2D |
| TAG | 0x32 |
| TCG | 0x36 |
| TGC | 0x39 |
| TTA | 0x3C |

| C | H | P |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 0 | 0 |

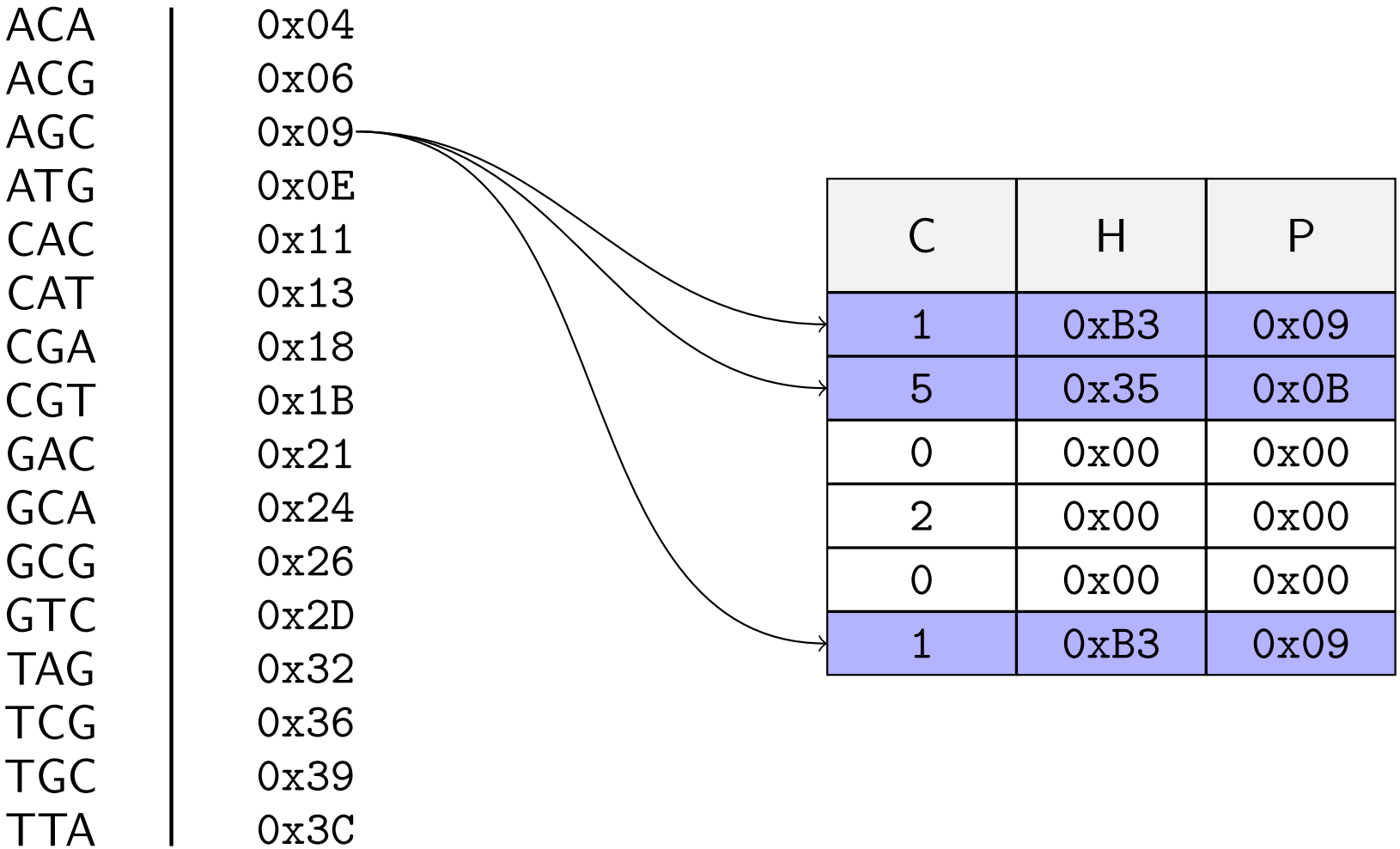
# IBLT: construction



# IBLT: construction

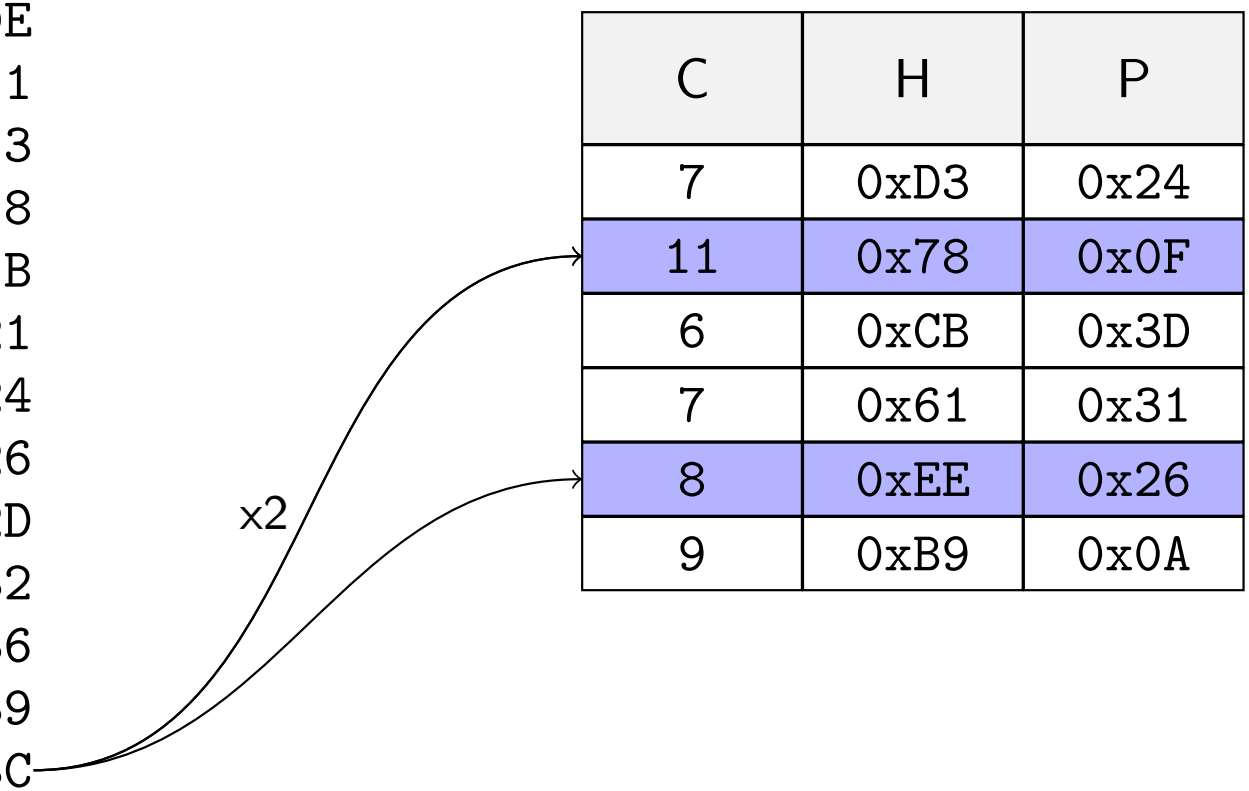


# IBLT: construction



# IBLT: construction

| | |
|---|---|
| ACA | 0x04 |
| ACG | 0x06 |
| AGC | 0x09 |
| ATG | 0x0E |
| CAC | 0x11 |
| CAT | 0x13 |
| CGA | 0x18 |
| CGT | 0x1B |
| GAC | 0x21 |
| GCA | 0x24 |
| GCG | 0x26 |
| GTC | 0x2D |
| TAG | 0x32 |
| TCG | 0x36 |
| TGC | 0x39 |
| TTA | 0x3C |




| C | H | P |
|---|---|---|
| 7 | 0xD3 | 0x24 |
| 11 | 0x78 | 0x0F |
| 6 | 0xCB | 0x3D |
| 7 | 0x61 | 0x31 |
| 8 | 0xEE | 0x26 |
| 9 | 0xB9 | 0x0A |

# IBLT: difference

| C | H | P |
|---|---|---|
| 7 | 0xD3 | 0x24 |
| 11 | 0x78 | 0x0F |
| 6 | 0xCB | 0x3D |
| 7 | 0x61 | 0x31 |
| 8 | 0xEE | 0x26 |
| 9 | 0xB9 | 0x0A |

- subtract a modified version of input $A$, called $B$ from $T$
  - $B$ obtained from $A$ by replacing CAC with CGC

# IBLT: peeling

The sketch is now peelable:

| C | H | P |
|---|---|---|
| 0 | 0x78 | 0x08 |
| -1 | 0x6E | 0x19 |
| 0 | 0x00 | 0x00 |
| -1 | 0x6E | 0x19 |
| 0 | 0x00 | 0x00 |
| 2 | 0x00 | 0x00 |

# IBLT: peeling

| C | H | P |
|---|---|---|
| 0 | 0x78 | 0x08 |
| -1 | 0x6E | 0x19 |
| 0 | 0x00 | 0x00 |
| -1 | 0x6E | 0x19 |
| 0 | 0x00 | 0x00 |
| 2 | 0x00 | 0x00 |

The sketch is now peelable:

- start by finding a bucket $i$ with $|T[i].C| = 1$

# IBLT: peeling

| C | H | P |
|---|---|---|
| 0 | 0x78 | 0x08 |
| -1 | 0x6E | 0x19 |
| 0 | 0x00 | 0x00 |
| -1 | 0x6E | 0x19 |
| 0 | 0x00 | 0x00 |
| 2 | 0x00 | 0x00 |

The sketch is now peelable:

- start by finding a bucket $i$ with $|T[i].C| = 1$

# IBLT: peeling

| C | H | P |
|---|---|---|
| 0 | 0x78 | 0x08 |
| -1 | 0x6E | 0x19 |
| 0 | 0x00 | 0x00 |
| -1 | 0x6E | 0x19 |
| 0 | 0x00 | 0x00 |
| 2 | 0x00 | 0x00 |

The sketch is now peelable:

- start by finding a bucket $i$ with $|T[i].C| = 1$
- extract k-mer stored in $T[i].P$: $0x19 \rightarrow$ CGC

# IBLT: peeling

| C | H | P |
|---|---|---|
| 0 | 0x78 | 0x08 |
| -1 | 0x6E | 0x19 |
| 0 | 0x00 | 0x00 |
| -1 | 0x6E | 0x19 |
| 0 | 0x00 | 0x00 |
| 2 | 0x00 | 0x00 |

The sketch is now peelable:

- start by finding a bucket $i$ with $|T[i].C| = 1$
- extract k-mer stored in $T[i].P$: $0x19 \rightarrow$ CGC
- delete CGC from $T$

# IBLT: peeling

| C | H | P |
|---|---|---|
| 1 | 0x16 | 0x11 |
| 0 | 0x00 | 0x00 |
| 0 | 0x00 | 0x00 |
| 0 | 0x00 | 0x00 |
| 0 | 0x00 | 0x00 |
| 2 | 0x00 | 0x00 |

The sketch is now peelable:

- start by finding a bucket $i$ with $|T[i].C| = 1$
- extract k-mer stored in $T[i].P$: $0x19 \rightarrow$ CGC
- delete CGC from $T$

# IBLT: peeling

| C | H | P |
|---|---|---|
| 1 | 0x16 | 0x11 |
| 0 | 0x00 | 0x00 |
| 0 | 0x00 | 0x00 |
| 0 | 0x00 | 0x00 |
| 0 | 0x00 | 0x00 |
| 2 | 0x00 | 0x00 |

The sketch is now peelable:

- start by finding a bucket $i$ with $|T[i].C| = 1$
- extract k-mer stored in $T[i].P$: $0x19 \rightarrow$ CGC
- delete CGC from $T$
- new peelable buckets can appear

# IBLT: peeling

| C | H | P |
|---|---|---|
| 1 | 0x16 | 0x11 |
| 0 | 0x00 | 0x00 |
| 0 | 0x00 | 0x00 |
| 0 | 0x00 | 0x00 |
| 0 | 0x00 | 0x00 |
| 2 | 0x00 | 0x00 |

The sketch is now peelable:

- start by finding a bucket $i$ with $|T[i].C| = 1$
- extract k-mer stored in $T[i].P$: $0x19 \rightarrow$ CGC
- delete CGC from $T$
- new peelable buckets can appear

# IBLT: peeling

| C | H | P |
|---|---|---|
| 1 | 0x16 | 0x11 |
| 0 | 0x00 | 0x00 |
| 0 | 0x00 | 0x00 |
| 0 | 0x00 | 0x00 |
| 0 | 0x00 | 0x00 |
| 2 | 0x00 | 0x00 |

The sketch is now peelable:

- start by finding a bucket $i$ with $|T[i].C| = 1$
- extract k-mer stored in $T[i].P$: $0x19 \rightarrow$ CGC
- delete CGC from $T$
- new peelable buckets can appear
- delete CAC from $T$

# IBLT: peeling

| C | H | P |
|---|---|---|
| 1 | 0x16 | 0x11 |
| 0 | 0x00 | 0x00 |
| 0 | 0x00 | 0x00 |
| 0 | 0x00 | 0x00 |
| 0 | 0x00 | 0x00 |
| 2 | 0x00 | 0x00 |

The sketch is now peelable:

- start by finding a bucket $i$ with $|T[i].C| = 1$
- extract k-mer stored in $T[i].P$: $0x19 \rightarrow$ CGC
- delete CGC from $T$
- new peelable buckets can appear
- delete CAC from $T$

# IBLT: peeling

| C | H | P |
|---|---|---|
| 0 | 0x00 | 0x00 |
| 0 | 0x00 | 0x00 |
| 0 | 0x00 | 0x00 |
| 0 | 0x00 | 0x00 |
| 0 | 0x00 | 0x00 |
| 0 | 0x00 | 0x00 |

The sketch is now peelable:

- start by finding a bucket $i$ with $|T[i].C| = 1$
- extract k-mer stored in $T[i].P$: $0x19 \rightarrow$ CGC
- delete CGC from $T$
- new peelable buckets can appear
- delete CAC from $T$
- {CAC,CGC} is the symmetric difference between $A$ and $B$

# IBLT: idea

Use two IBLTs instead of minHash sketches to compute Jaccard:

- build sketches $T_A$ and $T_B$ of size $m$ from $A$ and $B$

# IBLT: idea

Use two IBLTs instead of minHash sketches to compute Jaccard:

- build sketches $T_A$ and $T_B$ of size $m$ from $A$ and $B$
- subtract $T_B$ from $T_A$ to obtain $T_{AB}$

# IBLT: idea

Use two IBLTs instead of minHash sketches to compute Jaccard:

- build sketches $T_A$ and $T_B$ of size $m$ from $A$ and $B$
- subtract $T_B$ from $T_A$ to obtain $T_{AB}$
- retrieve the symmetric difference $(A \setminus B) \cup (B \setminus A)$ by peeling $T_{AB}$
  - elements in $(A \setminus B)$ can be recognized from $(B \setminus A)$ by the sign of their count field

# IBLT: idea

Use two IBLTs instead of minHash sketches to compute Jaccard:

- build sketches $T_A$ and $T_B$ of size $m$ from $A$ and $B$
- subtract $T_B$ from $T_A$ to obtain $T_{AB}$
- retrieve the symmetric difference $(A \setminus B) \cup (B \setminus A)$ by peeling $T_{AB}$
  - elements in $(A \setminus B)$ can be recognized from $(B \setminus A)$ by the sign of their count field
- compute Jaccard as

$$J(A, B) = \frac{|A| - |A \setminus B|}{|A| + |B \setminus A|} = \frac{|B| - |B \setminus A|}{|B| + |A \setminus B|}.$$

# IBLT: idea

Use two IBLTs instead of minHash sketches to compute Jaccard:

- build sketches $T_A$ and $T_B$ of size $m$ from $A$ and $B$
- subtract $T_B$ from $T_A$ to obtain $T_{AB}$
- retrieve the symmetric difference $(A \setminus B) \cup (B \setminus A)$ by peeling $T_{AB}$
  - elements in $(A \setminus B)$ can be recognized from $(B \setminus A)$ by the sign of their count field
- compute Jaccard as

$$J(A, B) = \frac{|A| - |A \setminus B|}{|A| + |B \setminus A|} = \frac{|B| - |B \setminus A|}{|B| + |A \setminus B|}.$$

- All previous methods based on IBLTs left $B$ unsketched
  - inspired by [Porat and Lipsky, 2007] we obtain $T_{AB}$ by doing $T_A - T_B$
  - $T_A$ and $T_B$ must have the same number of buckets and the same hash functions

# Reducing space - lightweight IBLTs

Hash field $H$ is used to detect false peelable buckets

- $|T[i].C| = 1$ can arise due to unrelated $k$-mer colliding into the same bucket
- A false peelable bucket is detected whenever $h_e(T[i].P) \neq T[i].H$

# Reducing space - lightweight IBLTs

Hash field $H$ is used to detect false peelable buckets

- $|T[i].C| = 1$ can arise due to unrelated $k$-mer colliding into the same bucket
- A false peelable bucket is detected whenever $h_e(T[i].P) \neq T[i].H$

We remove $H$ to save space:

# Reducing space - lightweight IBLTs

Hash field $H$ is used to detect false peelable buckets

- $|T[i].C| = 1$ can arise due to unrelated $k$-mer colliding into the same bucket
- A false peelable bucket is detected whenever $h_e(T[i].P) \neq T[i].H$

We remove $H$ to save space:

- For a potential peelable bucket at position $i$ we check $i \in h_1(T[i].P), \cdots, h_r(T[i].P)$
- Failure probability depends on sketch size $m$

# Reducing space - sampling

When comparing a mutated sequence to its original:

- Up-to $k$ new $k$-mers for each mutation
- Symmetric difference up to $k$ times bigger than what would be necessary

# Reducing space - sampling

When comparing a mutated sequence to its original:

- Up-to $k$ new $k$-mers for each mutation
- Symmetric difference up to $k$ times bigger than what would be necessary

Sampling $k$-mers *before* inserting them into an IBLT can help.

# Reducing space - sampling

When comparing a mutated sequence to its original:

- Up-to $k$ new $k$-mers for each mutation
- Symmetric difference up to $k$ times bigger than what would be necessary

Sampling $k$-mers *before* inserting them into an IBLT can help.

Three different possibilities:

1. hash-based sampling
2. minimizers: biased estimators of Jaccard similarity [Belbasi, Blanca, Harris, Koslicki, and Medvedev, 2022]
3. syncmers

# Reducing space - sampling

When comparing a mutated sequence to its original:

- Up-to $k$ new $k$-mers for each mutation
- Symmetric difference up to $k$ times bigger than what would be necessary

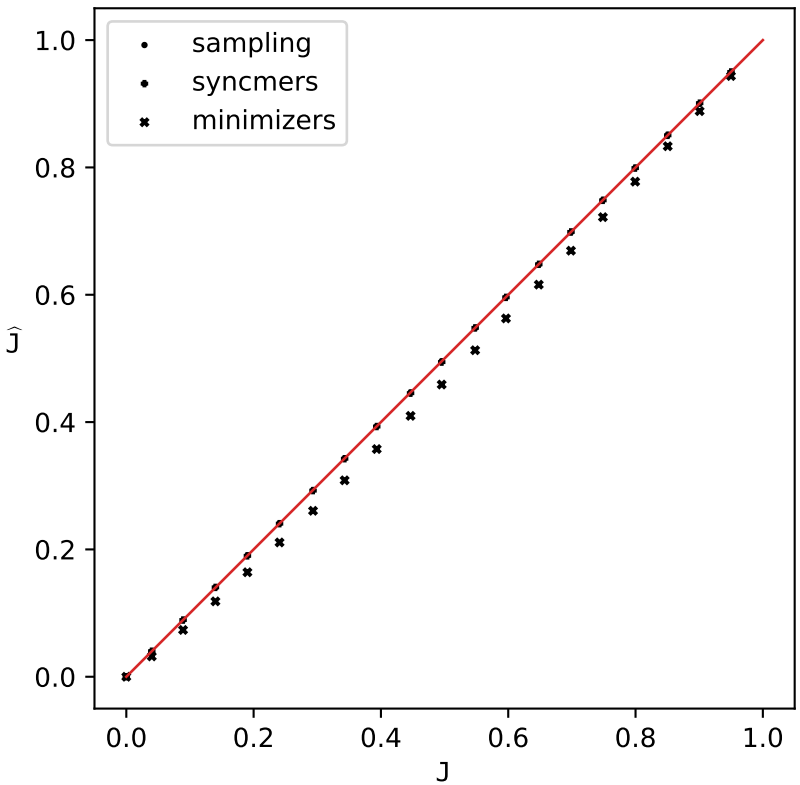
Sampling $k$-mers *before* inserting them into an IBLT can help.

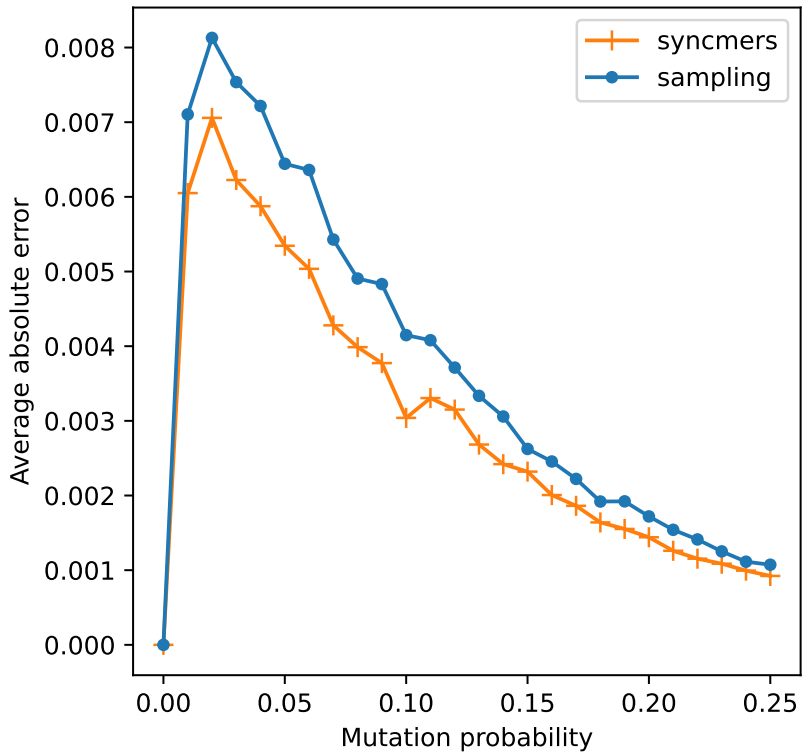
Three different possibilities:

1. hash-based sampling
2. minimizers: biased estimators of Jaccard similarity [Belbasi, Blanca, Harris, Koslicki, and Medvedev, 2022]
3. syncmers

Which one to choose?

# Hash-based sampling vs minimizers vs syncmers




All methods use the same sampling rate $\nu$ by choosing minimizer and syncmer lengths accordingly. Each point is the average over 500 trials using simulated sequences of length $L = 10\text{K}$.

# IBLT dimensioning with syncmers

Assuming

- Sequences of length $L$
- mutation rate $p_m$
- $k$ large enough to have unique $k$-mers
- syncmer parameter $z$

# IBLT dimensioning with syncmers

Assuming

- Sequences of length $L$
- mutation rate $p_m$
- $k$ large enough to have unique $k$-mers
- syncmer parameter $z$

we have

- $2k$ new $k$-mer per mutation in the symmetric difference
- $p_m L$ mutations
- syncmer density $\approx \frac{2}{k-z+1}$ [Edgar, 2021]

# IBLT dimensioning with syncmers

Assuming

- Sequences of length $L$
- mutation rate $p_m$
- $k$ large enough to have unique $k$-mers
- syncmer parameter $z$

we have

- $2k$ new $k$-mer per mutation in the symmetric difference
- $p_m L$ mutations
- syncmer density $\approx \frac{2}{k-z+1}$ [Edgar, 2021]

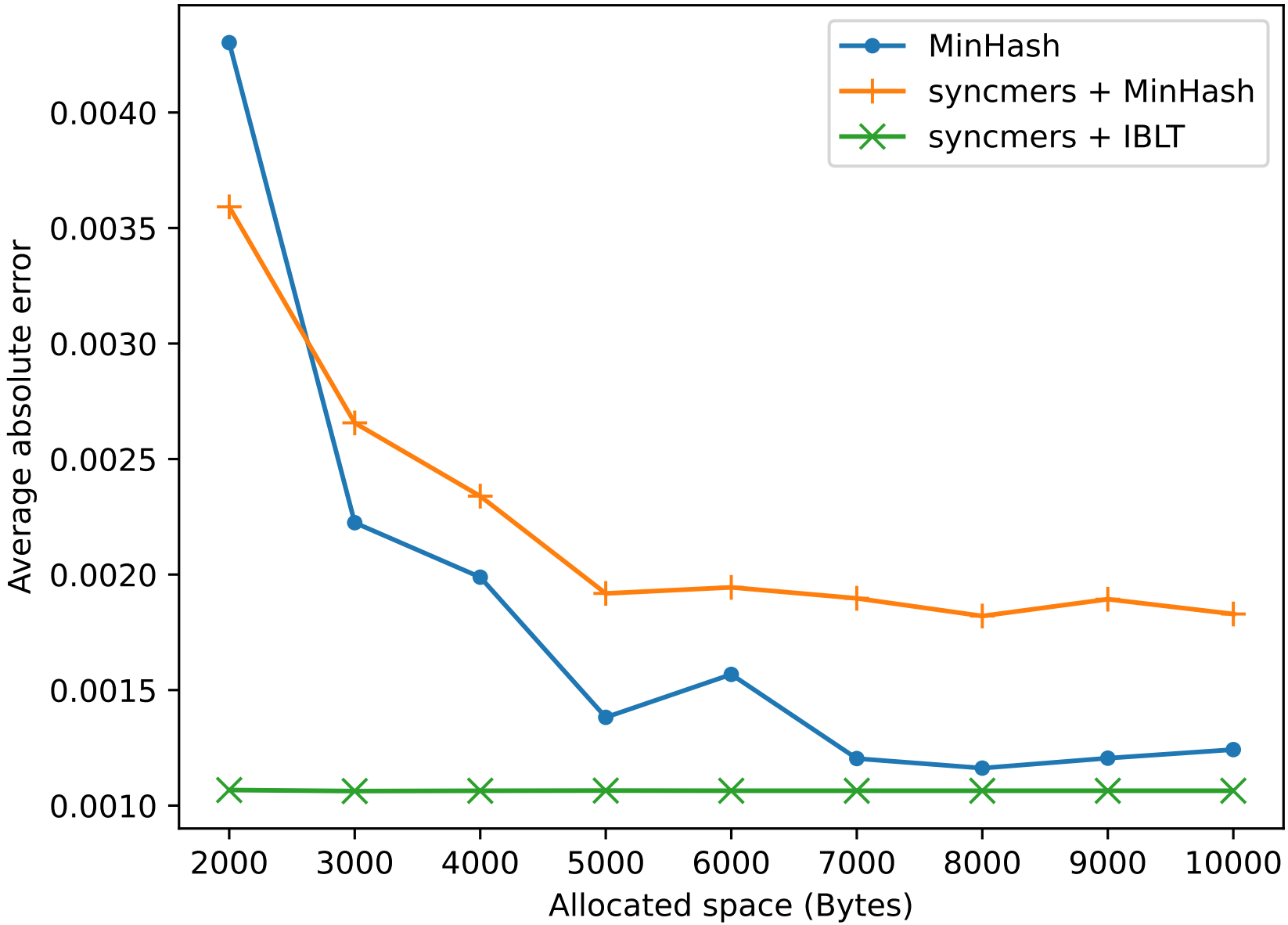
Then the expected number $n$ in the symmetric difference of syncmer sets is:

$$n = \frac{4kp_mL}{k-z+1}$$

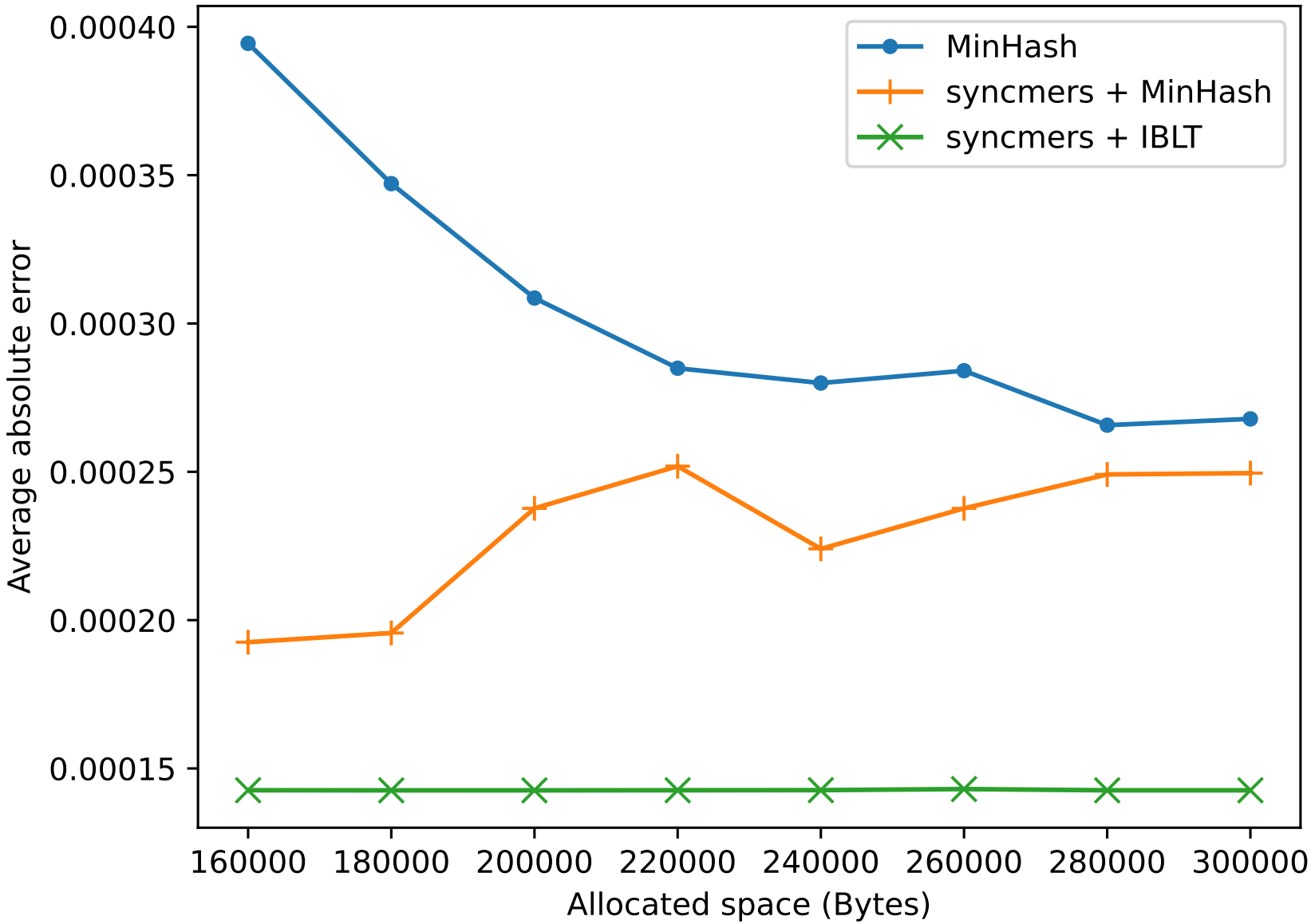
# Results

1. minHash comparisons on:
   - Subsample of 50 *SARS-CoV-2* genomes.
   - Subsample of 28 *Streprococcus Pneumoniae*. All sequences have pairwise mutation rates $< 0.0005$

# minHash vs IBLTs on *SARS-CoV-2* with $k = 15$, $z = 4$



# minHash vs IBLTs on *S.Pneumoniae* with $k = 15$, $z = 4$



# Conclusion

When sequences are of high similarity:

1. $k$-mer set difference can be found without storing the whole sets
2. $T_{AB} = T_A - T_B$ instead of $T_A B = T_A - B$
3. Hash field $H$ can be removed in order to save space
4. Syncmers + IBLTs are better than minHash sketches for Jaccard estimation

# Additional work

Further results not presented here include:

- Hash-based sampling + syncmers $\rightarrow$ even smaller sketches + (errors)
- Retrieval of supersets $k$-mer set symmetric differences by tracking only $n' < kn$ differences

Preprint: "Efficient reconciliation of genomic datasets of high similarity" at doi.org/10.1101/2022.06.07.495186

# Implementation

IBLTs implemented in “km-peeler” at
`https://github.com/yhhshb/km-peeler`

# Future work

- Make use of the $k$-mers in the symmetric difference:

# Future work

- Make use of the $k$-mers in the symmetric difference:
  - Locally assemble them ?
  - Use them as seeds ?

# Future work

- Make use of the $k$-mers in the symmetric difference:
  - Locally assemble them ?
  - Use them as seeds ?
- Improve km-peeler:
  - Store multiplicities together with $k$-mers
  - Variable-length payload fields $P$

# Future work

- Make use of the $k$-mers in the symmetric difference:
  - Locally assemble them ?
  - Use them as seeds ?
- Improve km-peeler:
  - Store multiplicities together with $k$-mers
  - Variable-length payload fields $P$
- Other applications
  - Fast VCF difference computation for genotyping
  - Efficient unique substring identification

# References I


Mahdi Belbasi, Antonio Blanca, Robert S. Harris, David Koslicki, and Paul Medvedev. The minimizer jaccard estimator is biased and inconsistent. *bioRxiv*, 2022. doi: 10.1101/2022.01.14.476226. URL https://www.biorxiv.org/content/early/2022/01/17/2022.01.14.476226.

Robert Edgar. Syncmers are more sensitive than minimizers for selecting conserved k-mers in biological sequences. *PeerJ*, 9:e10805, February 2021. ISSN 2167-8359. doi: 10.7717/peerj.10805. URL https://peerj.com/articles/10805.

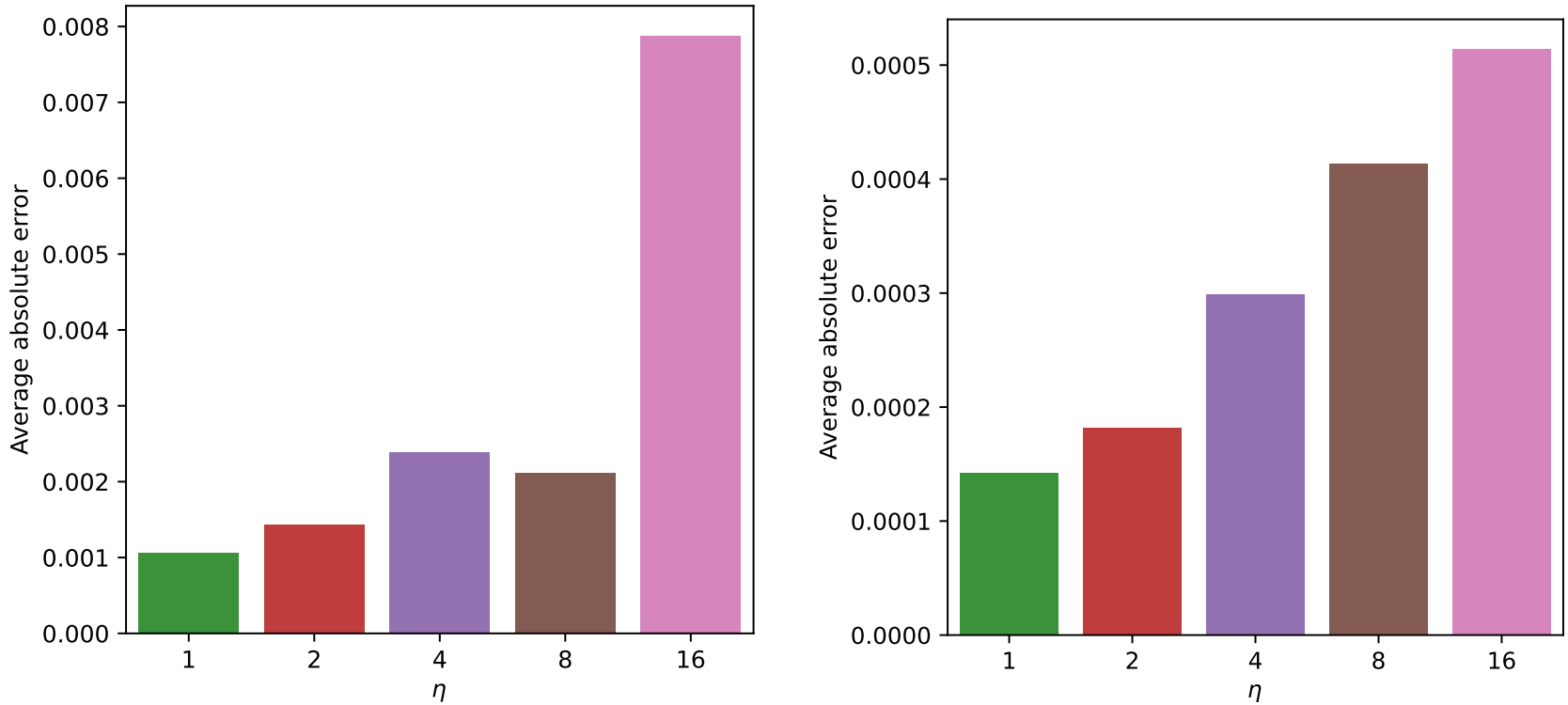
Michael T. Goodrich and Michael Mitzenmacher. Invertible Bloom lookup tables, 2011. URL https://arxiv.org/abs/1101.2245.

Brian D. Ondov, Todd J. Treangen, Páll Melsted, Adam B. Mallonee, Nicholas H. Bergman, Sergey Koren, and Adam M. Phillippy. Mash: fast genome and metagenome distance estimation using MinHash. *Genome Biology*, 17(1):132, June 2016. ISSN 1474-760X. doi: 10.1186/s13059-016-0997-x. URL https://doi.org/10.1186/s13059-016-0997-x.

# References II

Ely Porat and Ohad Lipsky. Improved sketching of hamming distance with error correcting. In Bin Ma and Kaizhong Zhang, editors, *Combinatorial Pattern Matching*, pages 173–182, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. ISBN 978-3-540-73437-6.

# Sampling syncmers before IBLT insertion

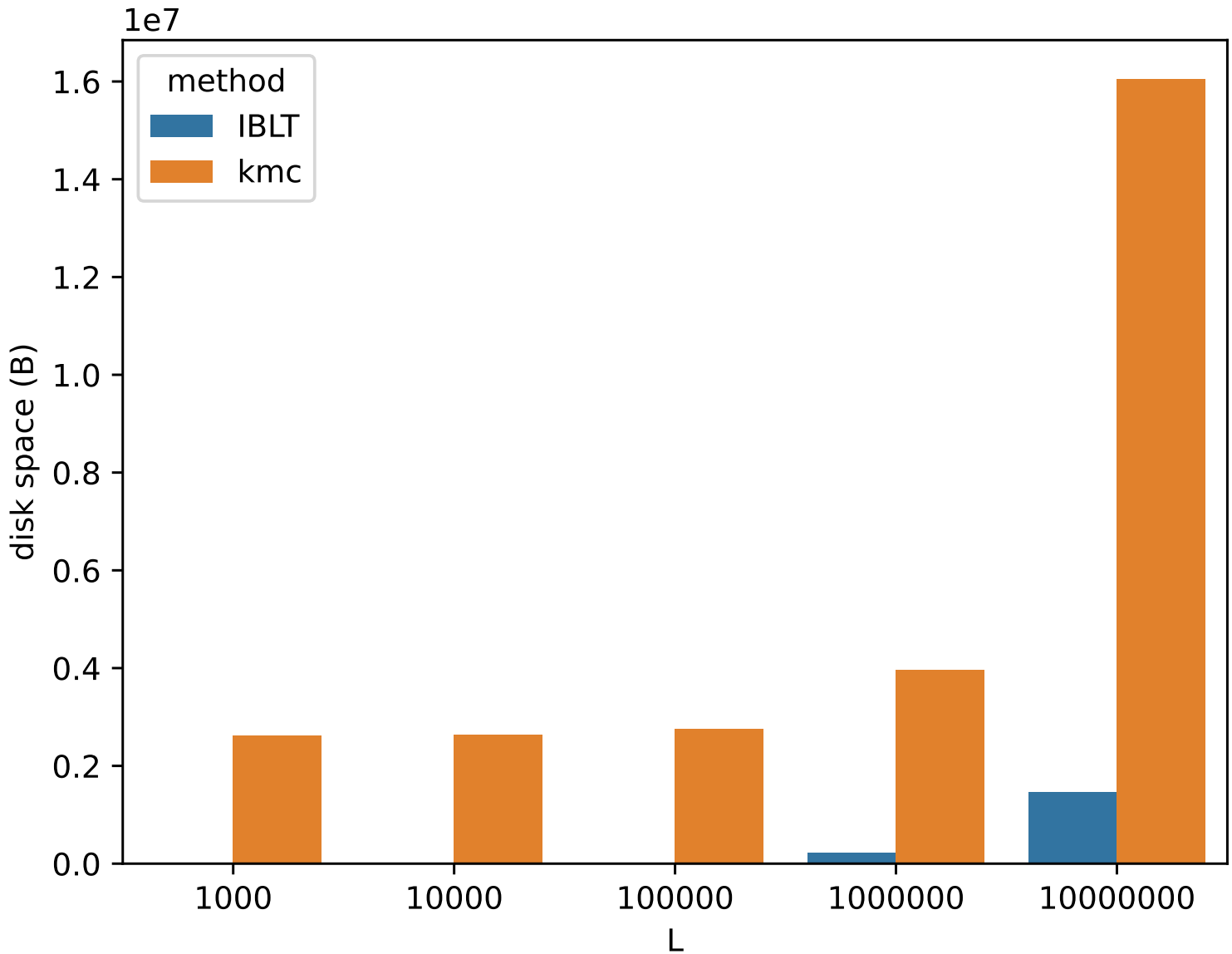



(a) `covid` dataset

(b) `spneu` dataset

Figure: Effect of sampling syncmers before IBLT insertion on the average absolute error. $\eta$ is the compression rate ($1/\nu$) used for sampling syncmer sets before IBLT insertion. $\eta = 1$ means no sampling (full syncmer sets).

# IBLTs vs KMC - mutation probability = 0.001



# IBLTs vs KMC - mutation probability = 0.01