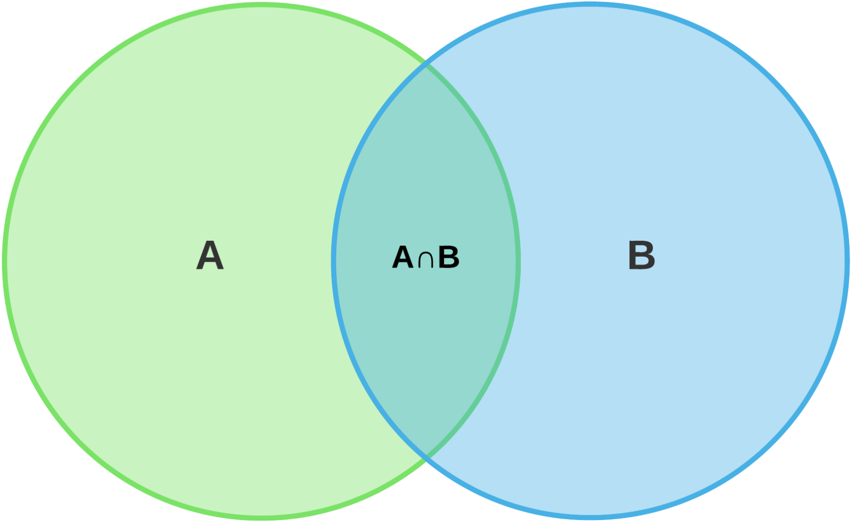
# Bcash: Best compressible hash

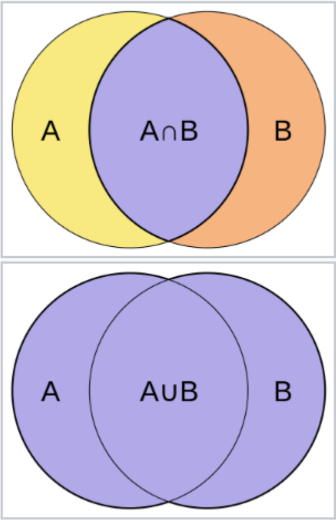
**Antoine Limasset**, Yoshihiro Shibuya , Rayan Chikhi and Gregory Kucherov

January 30, 2020

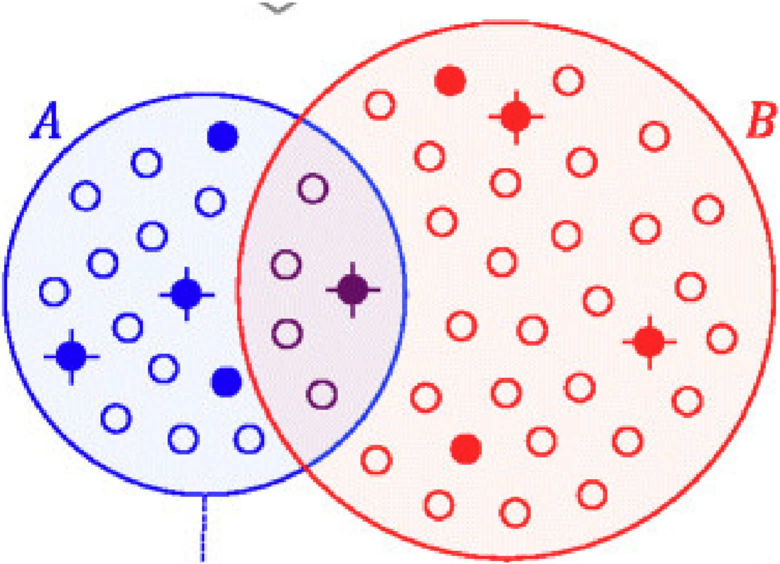
# Set similarity



# Jaccard coefficient




Intersection and union of two sets A and B


A
B

# Three flavor of minhash

## H hash functions

Compute H hash functions on each key
$\mathcal{O}(nH)$

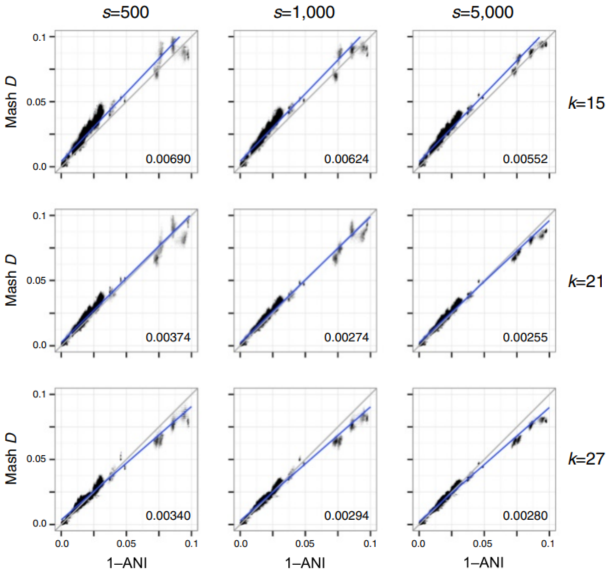
## H minimum hashes

One hash function, keep the H smallest hashes
$\mathcal{O}(n \log H)$

## H partitions

One hash function, split the hashes into H partition according to their first bit.
Each partition keep its smallest hash
$\mathcal{O}(n)$

# Application to bioinformatics: MASH



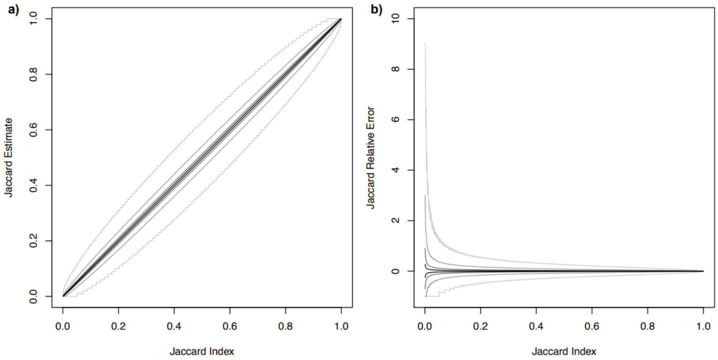



**Figure S1. Absolute and relative error bounds for Mash Jaccard estimates given various sketch sizes.** Increasing sketch sizes are progressively shaded from *s*=100 (light gray), *s*=1,000, *s*=10,000, and *s*=100,000 (black). Upper and lower bounds are drawn using the binomial inverse cumulative distribution function, with the same parameters from equation 8, such that for a given Jaccard index there is a 0.99 probability that the corresponding Jaccard estimate **(a)** or relative error **(b)** will fall within the bounds. These plots illustrate that relative error can grow quite large when estimating small Jaccard values. Thus, large sketch sizes are recommended when comparing divergent sequences with few shared k-mers. These plots only illustrate the error of the Jaccard estimate, and are independent of k-

# Hash size

b bits hashes

**Pros**

Probability of collision: $\frac{1}{2^b}$

**Cons**

Sketch size: $H * b$ bits

$b = \mathcal{O}(\log n)$

# Minimizer evolution

## First hash

```
10100111 00000010 01010110 01110011 01011010 10111011 00110100 00000110
```

Found an inferior hash, we have a new minimizer

```
10100111 00000010 01010110 01110011 01011010 10111011 00110100 00000110
01100001 11011011 00011000 00011110 10111001 00111000 11001011 11110000
```

And so on,

```
10100111 00000010 01010110 01110011 01011010 10111011 00110100 00000110

01100001 11011011 00011000 00011110 10111001 00111000 11001011 11110000

01011000 11010001 00001001 00101101 10100011 00011101 01000101 01110001
```

# Minimizer evolution

```
10100111 00000010 01010110 01110011 01011010 10111011 00110100 00000110

01100001 11011011 00011000 00011110 10111001 00111000 11001011 11110000

01011000 11010001 00001001 00101101 10100011 00011101 01000101 01110001

00101101 11011000 11000111 10000111 01001111 10111000 10101001 01010110
```

# Minimizer evolution

```
10100111 00000010 01010110 01110011 01011010 10111011 00110100 00000110

01100001 11011011 00011000 00011110 10111001 00111000 11001011 11110000

01011000 11010001 00001001 00101101 10100011 00011101 01000101 01110001

00101101 11011000 11000111 10000111 01001111 10111000 10101001 01010110

00001010 01111011 01001100 00000110 10000011 01011011 11111010 11000110
```

# Minimizer evolution

```
10100111 00000010 01010110 01110011 01011010 10111011 00110100 00000110

01100001 11011011 00011000 00011110 10111001 00111000 11001011 11110000

01011000 11010001 00001001 00101101 10100011 00011101 01000101 01110001

00101101 11011000 11000111 10000111 01001111 10111000 10101001 01010110

00001010 01111011 01001100 00000110 10000011 01011011 11111010 11000110

… hundred  steps later

00000000 11100111 11100011 00000010 10000101 00100101 01110110 01001010

… hundred thousand steps later

00000000 00000000 10111011 10000100 00000010 10110101 10010100 00001100
```

# Hyperminhash observation

```
10100111 00000010 01010110 01110011 01011010 10111011 00110100 00000110
(0,0100111000)

01100001 11011011 00011000 00011110 10111001 00111000 11001011 11110000
(1,1000011101)

0101100011010001 00001001 00101101 10100011 00011101 01000101 01110001
(1,0110001101)

00101101 11011000 11000111 10000111 01001111 10111000 10101001 01010110
(2,0110111011)

00001010 01111011 01001100 00000110 10000011 01011011 11111010 11000110
(4,0100111101)

… hundred  steps later

00000000 11100111 11100011 00000010 10000101 00100101 01110110 01001010
(8,1100111111)

… hundred thousand steps later

00000000 00000000 10111011 10000100 00000010 10110101 10010100 00001100
(16,0111011100)
```

# HyperMinHash: MinHash in LogLog space

```
00000000 00000000 10111011 10000100 00000010 10110101 10010100 00001100
(16,0111011100)
```

A hyperminhash fingerprint is a hyperloglog fingerprint (6 bits) and a constant size finger print (10bit)

Lossy compression from $O(\log n)$ to $O(\log \log n)$

Allow cardinality estimation and unions estimation using the hyperloglog fingerprint

# Main Ideas

1. We can compress minimizers using the fact that they are **selected** among a large number of hashes
2. Minhash work with any order relation (minimum,maximum,...)
3. We could select minimizers to be **compressible**

# Example: optimize run length

We select the hash with the minimal amount of bit flip

| | |
|---|---|
| 1111111 00 1 00 1 00 11111 0 1 0 1 0 1 0 11 0 11 | 16 flips |
| 00000000 11111 0000000 1 000000000 1 0 | 6 flips |

# Terminal time!

```
init score:  16
11111110010010011111010101011011
best score:  15 step:  1
01110100000100101111100010110001
best score:  14 step:  2
11011111100010011100110100110011
best score:  12 step:  22
01110000001001100010011111011100
best score:  10 step:  29
01101111110101100000001111111100
best score:  9 step:  39
11000000001110001111111110101110
best score:  8 step:  363
11111000111110100010111111111111
best score:  7 step:  1432
11111111111110001110000101111000
best score:  6 step:  1666
00000000111110000000010000000010
```

```
init score:  14
10111010000001100000100010111101
best score:  10 step:  4
00011000000110111000111001110000
best score:  8 step:  101
11010011111110000000000111111101
best score:  7 step:  262
11111110011000000000110000000110
best score:  6 step:  492
00001111100010000000001111111100
best score:  5 step:  11088
11100110001111111111111110000000
best score:  3 step:  19419
00000000011000011111111111111111
```

# Example: optimize amount of 0

We select the hash with the minimal amount of 1

```
init score:  20
11110001010000001010101101010110101
best score:  18 step:  0
00011001001111001001101101111110
best score:  11 step:  2
00010100000101010000100000011011101
best score:  10 step:  35
00000011010011010010010001000100
best score:  9 step:  54
10000110000100100010010000011001
best score:  7 step:  145
00000110010110000000000010001001
best score:  6 step:  4383
00000100001011000000100001000000000
best score:  5 step:  10587
10010000001100000100000000000000
best score:  4 step:  24951
00000000000000010001100010000000
```

```
init score:  18
01001010110011110000100110110110
best score:  11 step:  0
00100000101110000000111000100101 0
best score:  10 step:  12
01001001000000000100101100101010 1
best score:  9 step:  44
00011000001101011010000000000001000
best score:  8 step:  808
10010010100000000000010100000010010
best score:  7 step:  1336
01111000000000010000000010000000011
best score:  6 step:  2611
00010001111000000000010000000000000
best score:  5 step:  4968
00100000001001000010000000000000011
best score:  4 step:  21708
00100000000000001001000000000000010
```

### H-partition sketching of 1 billion 32bits hashes

Gzip compression of the sketches using different strategies

1. minimizing value (vanilla minhash)
2. minimizing amount of 1
3. minimizing amount of flips

A minimizer is chosen among 100 hashes (on average)

| Strategy | Size | Compression ratio |
|---|---|---|
| IDENTITY | 37,460,166 | 1.068 |
| NUMBER 1 | 35,242,423 | **1.135** |
| NUMBER FLIP | 35,557,655 | 1.125 |

The uncompressed sketch file is 40,000,000 bytes

A minimizer is chosen among 1000 hashes (on average)

| Strategy | Size | Compression ratio |
|---|---|---|
| IDENTITY | 3,422,813 | 1.169 |
| NUMBER 1 | 3,198,364 | **1.251** |
| NUMBER FLIP | 3,242,664 | 1.234 |

The uncompressed sketch file is 4,000,000 bytes

# 100,000 MINIMIZERS

A minimizer is chosen among 10,000 hashes (on average)

| Strategy | Size | Compression ratio |
|---|---|---|
| IDENTITY | 319,662 | 1.251 |
| NUMBER 1 | 295,284 | **1.355** |
| NUMBER FLIP | 299,484 | 1.336 |

The uncompressed sketch file is 400,000 bytes

# 10,000 MINIMIZERS

A minimizer is chosen among 100,000 hashes (on average)

| Strategy | Size | Compression ratio |
|---|---|---|
| IDENTITY | 28,762 | 1.391 |
| NUMBER 1 | 26,796 | **1.493** |
| NUMBER FLIP | 27,206 | 1.47026 |

The uncompressed sketch file is 40,000 bytes

# 1,000 MINIMIZERS

A minimizer is chosen among 1,000,000 hashes (on average)

| Strategy | Size | Compression ratio |
|---|---|---|
| IDENTITY | 2,557 | 1.564 |
| NUMBER 1 | 2,393 | **1.672** |
| NUMBER FLIP | 2,447 | 1.635 |

The uncompressed sketch file is 4,000 bytes

A minimizer is chosen among 10,000,000 hashes (on average)

| Strategy | Size | Compression ratio |
|---|---|---|
| IDENTITY | 280 | 1.429 |
| NUMBER 1 | 245 | **1.633** |
| NUMBER FLIP | 256 | 1.563 |

The uncompressed sketch file is 400 bytes

# Remark

1. Naive compression (gzip)
2. Naive selection
3. **Lossless** compression

A minimizer is chosen among 1,000,000 hashes (on average)

| Strategy | Size | Compression ratio |
|---|---|---|
| IDENTITY | 2,557 | 1.564 |
| NUMBER 1 | 2,393 | 1.672 |
| NUMBER FLIP | 2,447 | 1.635 |
| NUMBER FLIP+BITWISE RLE | 2,184 | **1.832** |

# 100 MINIMIZERS

A minimizer is chosen among 10,000,000 hashes (on average)

| Strategy | Size | Compression ratio |
|---|---|---|
| IDENTITY | 280 | 1.429 |
| NUMBER 1 | 245 | 1.633 |
| NUMBER FLIP | 256 | 1.563 |
| NUMBER FLIP+BITWISE RLE | 207 | **1.932** |

# Lossy compression examples

1. Encode the *n* first flip lengths
2. Encode the *n* first 1 positions
3. Encode the *n* longest flip lengths
4. Encode amount of 0

How to take into account collisions?

# Lossy compression pros and cons

1. Skip hard parts
2. Better control on the fields to compress
3. **Harder analysis**

### Hard question

Good measure to estimate compressibility of 4 bytes integer

### Easy question

How to compress the previous such integer sketch

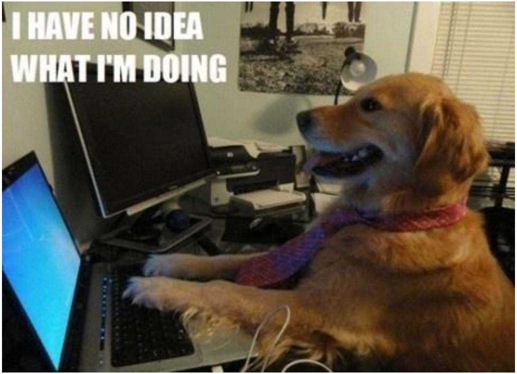
**Hard question**

How to compress a sketch

**Easy question**

How to optimize its compressibility by selecting minimizer

# Ideas/collaborations are welcome !




**Very easy to test and benchmark**

Benchmark available at `github.com/Malfoy/Bcash`
Write a score function and see how the sketch can be compressed!