Kraken 2

Improved metagenomic analysis with Kraken 2
Derrick E. Wood, Jennifer Lu & Ben Langmead
Genome Biol 20, 257 (2019)
https://doi.org/10.1186/s13059-019-1891-0

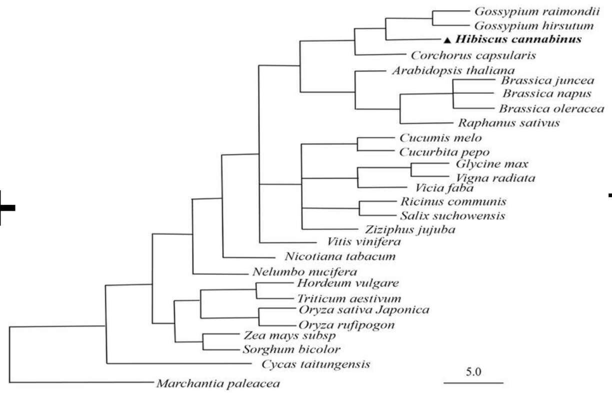
Konstantin Schütze
Advanced Bioinformatics 1
2021-09-15

Metagenomic Sequence Classification

Reads + Reference Genomes → Taxonomic assignment

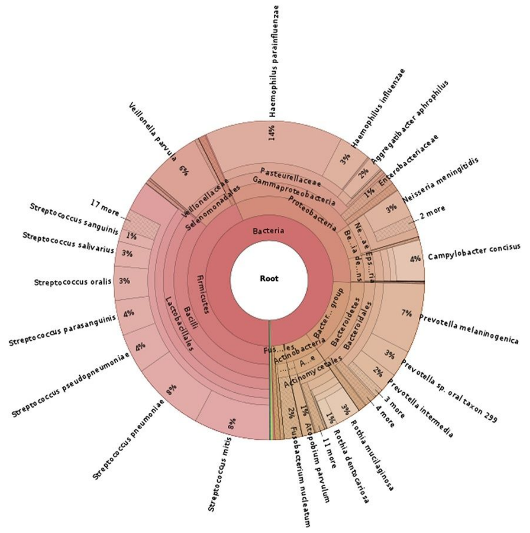
Illumina HiSeq 4000

+



A taxomy
https://doi.org/10.1038/s41598-018-30297-w

→

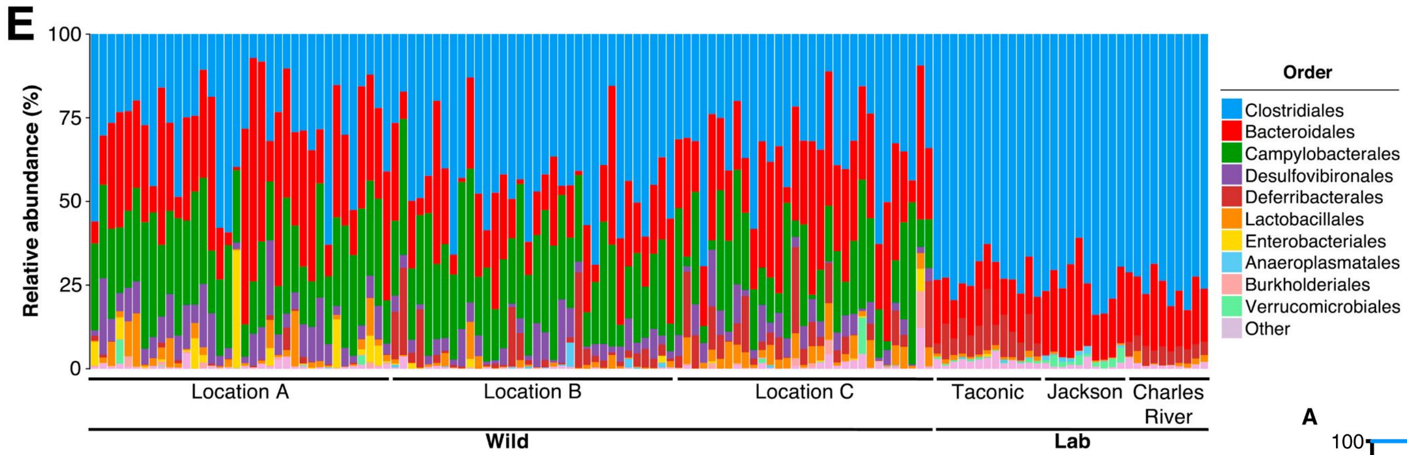


Taxonomic distribution of saliva microbiome (kraken 1)

Why do we even need that?

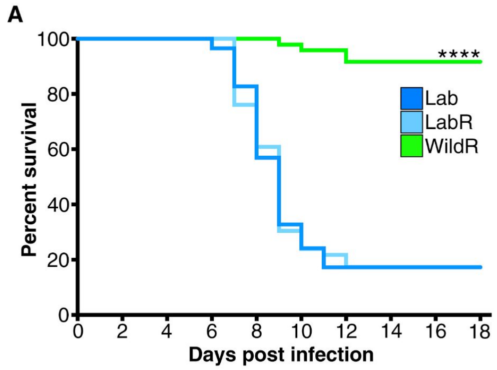
- Wild rat microbiome improves lab rat health
  https://doi.org/10.1016/j.cell.2017.09.016
- Zika virus evolution
  https://doi.org/10.1038/nature22402
- Finding the source of Pseudomonas aeruginosa infections in a hospital
  http://dx.doi.org/10.1136/bmjopen-2014-006278

Rat Microbiome Diversity



High diversity in wild rats

Low diversity in lab rats



Rosshart et al. (2017) https://doi.org/10.1016/j.cell.2017.09.016

Methods

1. **BLAST**: Search entire read
    - “Classic”
    - accurate
    - slow
2. **k-mers**: Split read into k-mers
    - faster
    - near-BLAST accuracy
3. **marker genes**: DB with only few genes (16S)
    - very fast
    - less accurate

Kraken 1

1. Build fast mapping reference genomes k-mers -> taxonomy id
    - Minimizer
    - Offset table
2. Query all k-mers from read
3. Assign read by majority assignment of k-mers

Minimizer (l-mer)

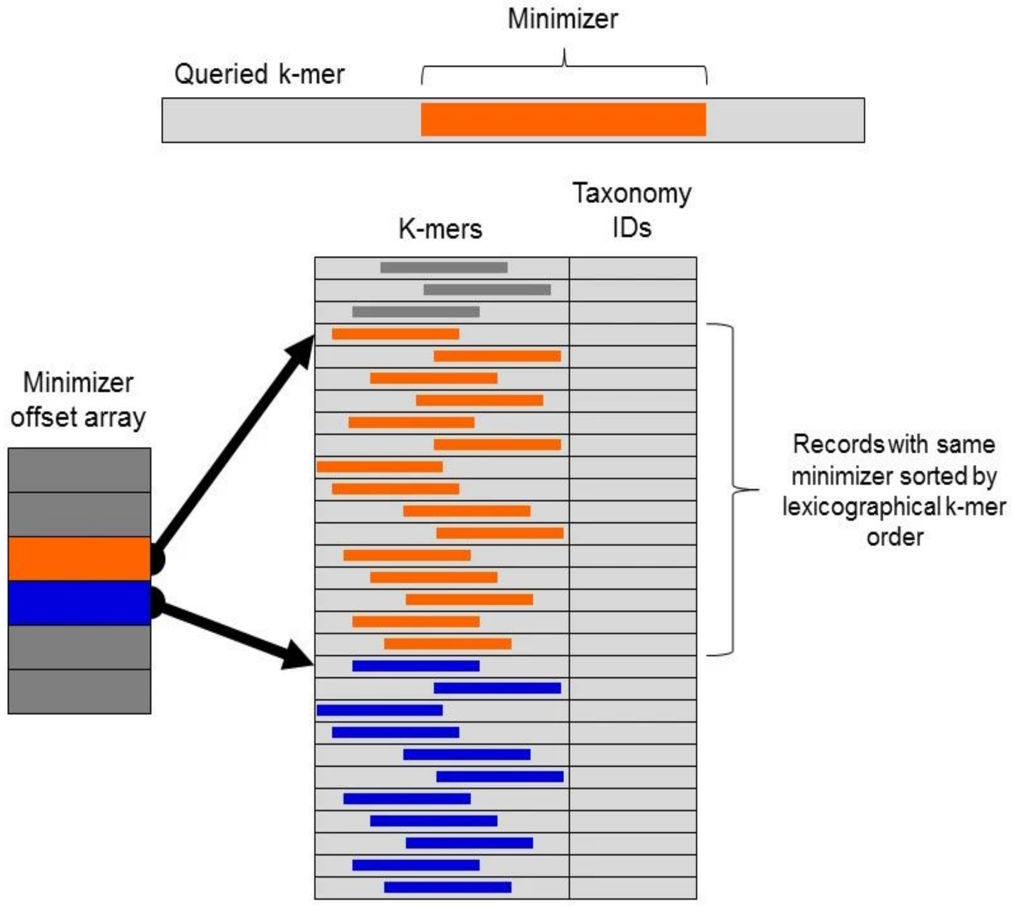
Minimizer: Lexicographically smallest l-mer inside the k-mer

```
ATGATCGTGCATCCATCAGTCATCAGTA <- initial k-mer

ATGATCGTGCATCCATCAGTCA
 TGATCGTGCATCCATCAGTCAT
  TATCGTGCATCCATCAGTCATC
   ATCGTGCATCCATCAGTCATCA <- minimizer¹ (l-mer)
    TCGTGCATCCATCAGTCATCAG
     CGTGCATCCATCAGTCATCAGT
      GTGCATCCATCAGTCATCAGTA
```

[1]ignoring XOR low complexity masking

Kraken 1

- Group k-mers by minimizers
- Minimizer offset array
  - 8×4^l bytes for l-mers
- Good for CPU caching


Minimizer
Queried k-mer
Taxonomy IDs
K-mers
Minimizer offset array
Records with same minimizer sorted by lexicographical k-mer order

Can we compress k-mers even more?

Spaced k-mers

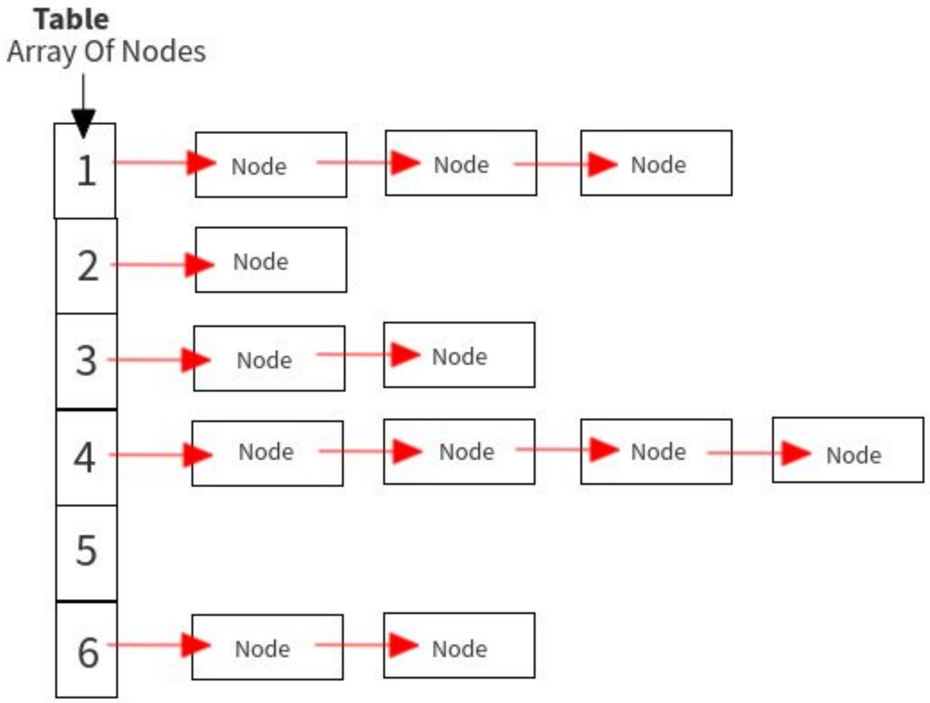
Mask *s* positions in l-mers (default: *s*=7)

```
AGTATCGTGCATCGATCAGTCATCAGTA  <- original k-mer
    TCGTGCATCGATCAGTCATCA     <- initial minimizer
    TCGTGCATCGATCAG-C-T-A     <- spaced minimizer
```

Kraken 2

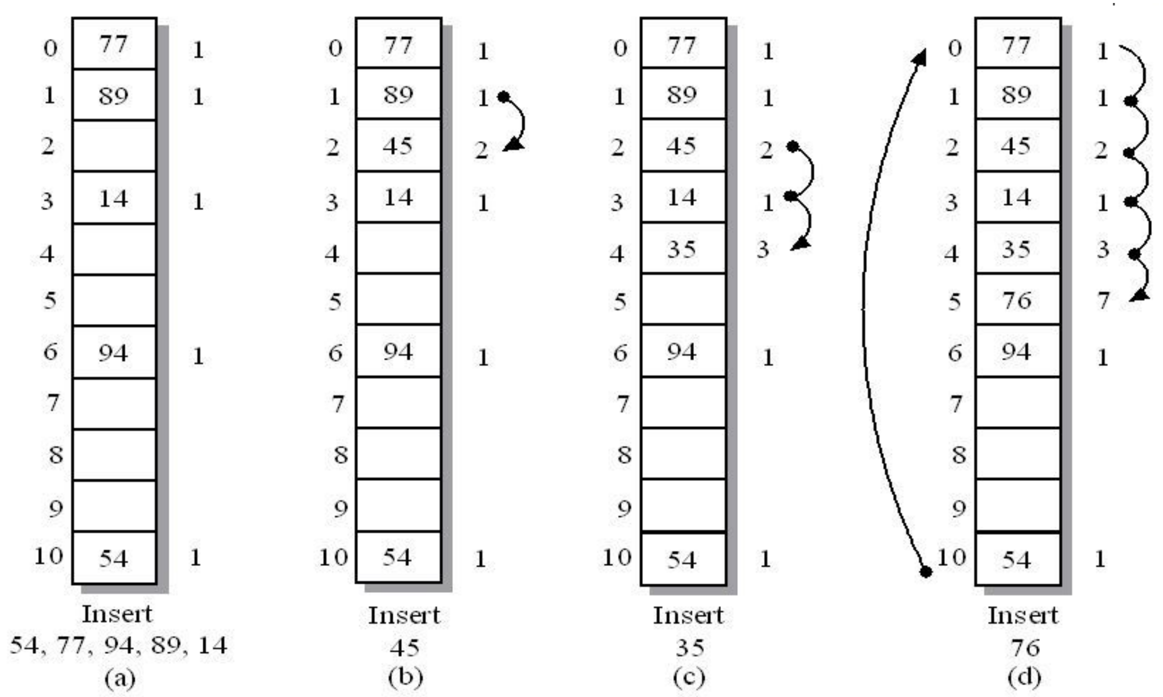
- Spaced k-mers
- k-mers share minimizers -> store only distinct minimizers
  - all k-mers -> few l-mers
- Only store hash of minimizer
- Defaults: k=35 l=31

Textbook Hash Maps



https://infondgndg91.blogspot.com/2020/07/java-hashmap-feat-linkedlist-tree-maybe.html

Compact Hash Table and Linear Probing



Load: 45%

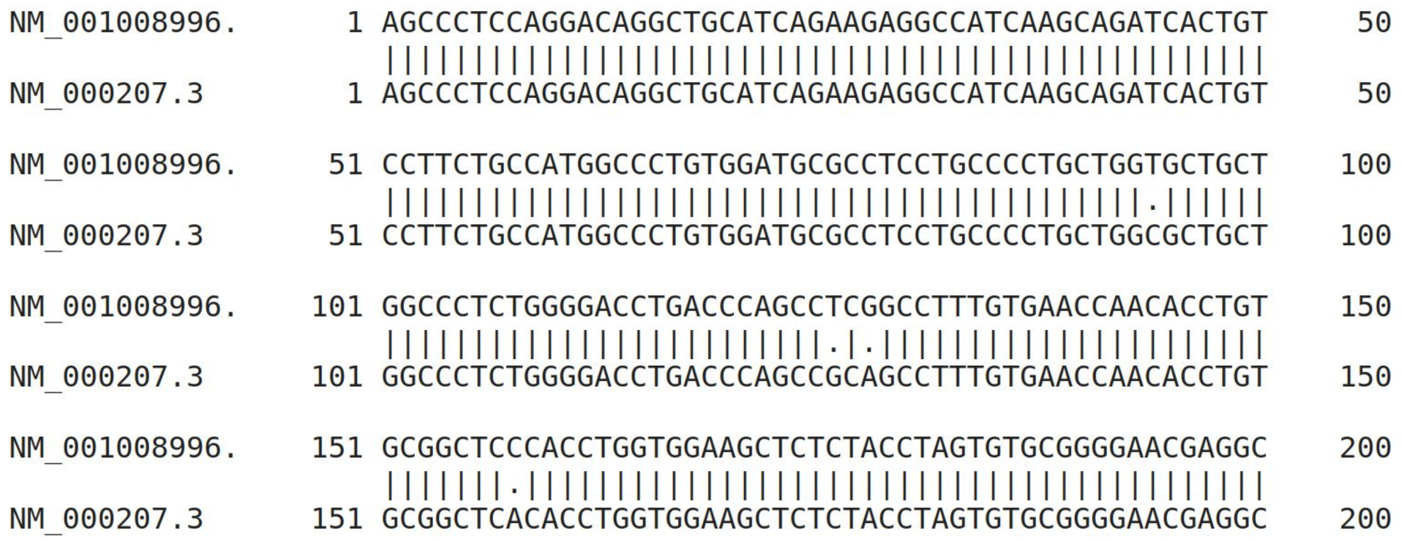
Load: 73%

Kraken 2: Memory Usage

- 4 bytes per entry: 15 bits hash, 17 bytes taxonomy id
- 70% load factor and D distinct minifier: **4D/0.7** bytes
- 85% less memory and 5x faster than Kraken 1

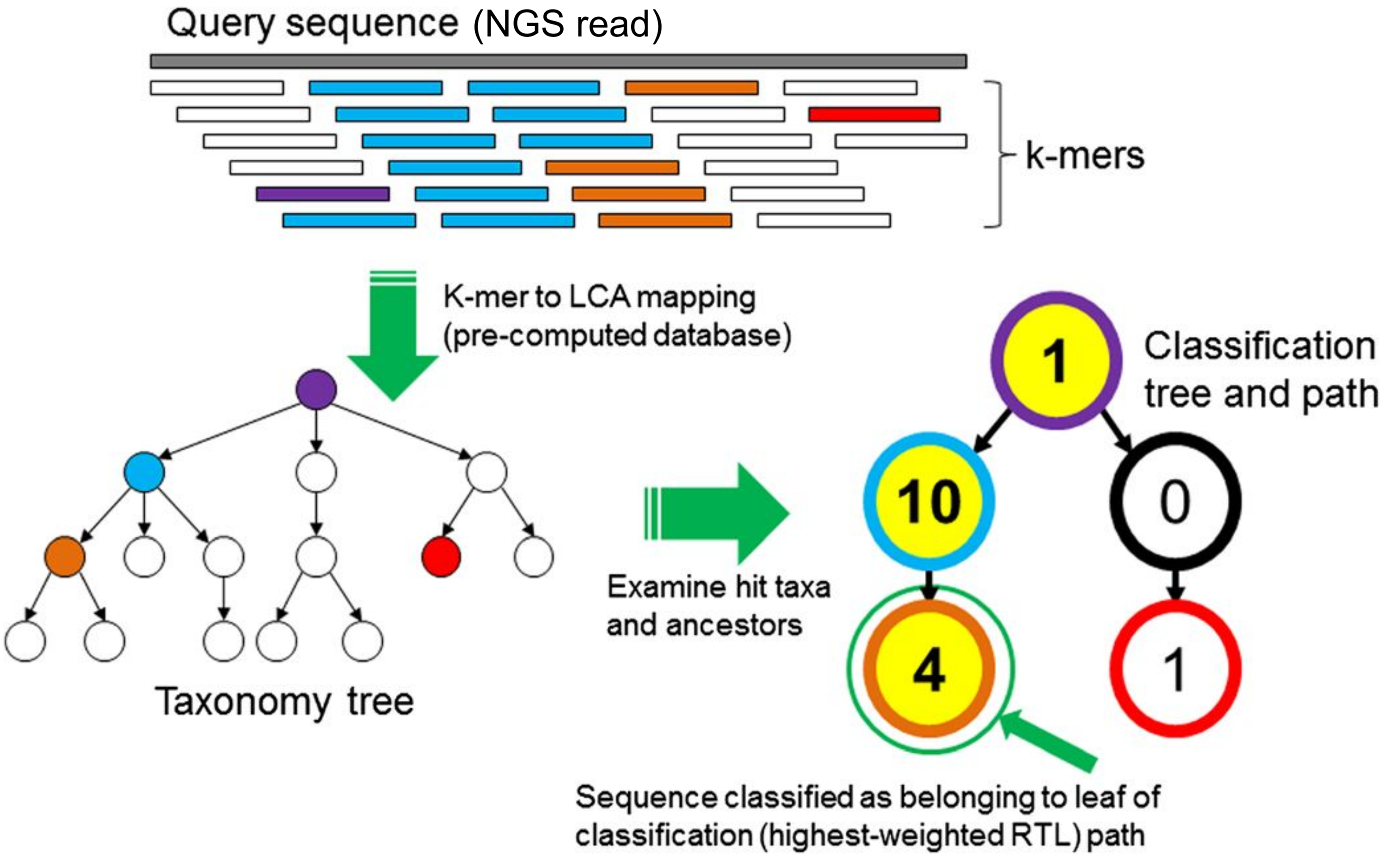
Building the Database

- When two minimizers collide we set them to the *Lowest Common Ancestor* (LCA) of both source species
- Example: Insulin in human and chimp -> LCA is Primate

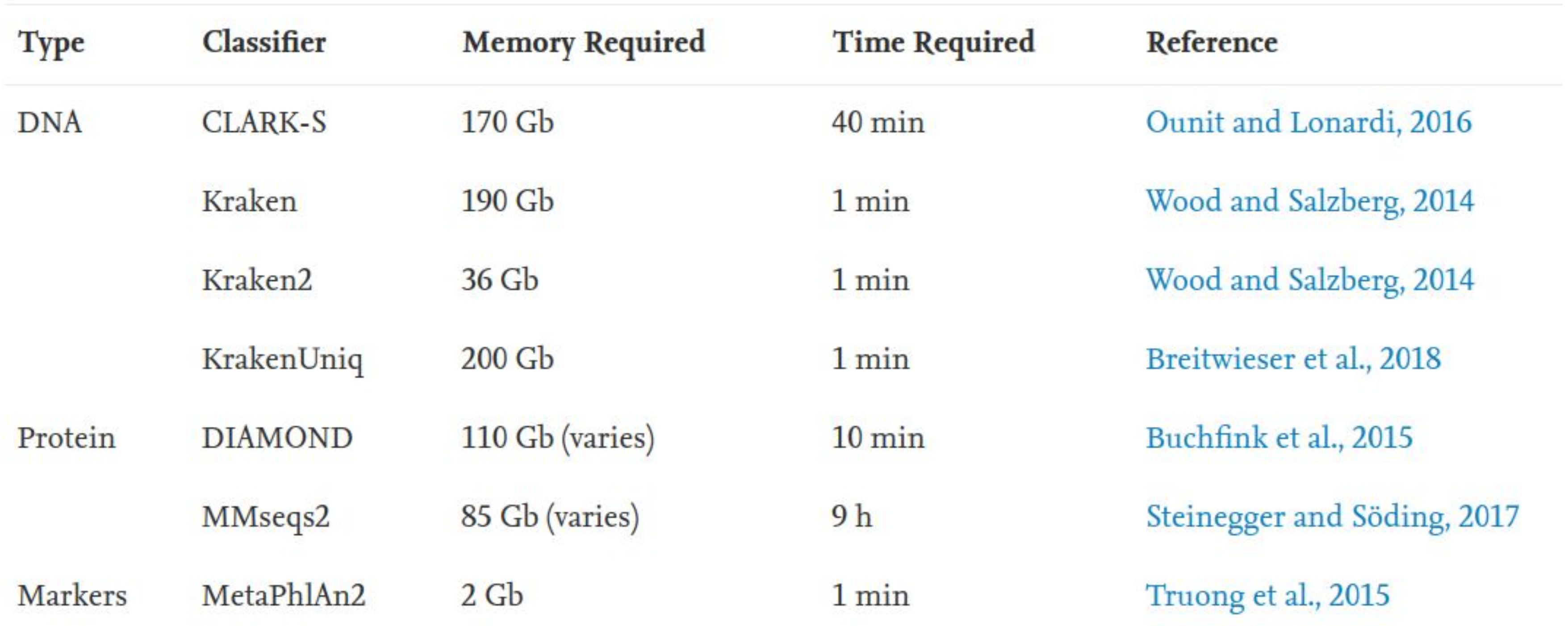


```
NM_001008996.      1 AGCCCTCCAGGACAGGCTGCATCAGAAGAGGCCATCAAGCAGATCACTGT     50
                     ||||||||||||||||||||||||||||||||||||||||||||||||||
NM_000207.3        1 AGCCCTCCAGGACAGGCTGCATCAGAAGAGGCCATCAAGCAGATCACTGT     50

NM_001008996.     51 CCTTCTGCCATGGCCCTGTGGATGCGCCTCCTGCCCCTGCTGGTGCTGCT    100
                     |||||||||||||||||||||||||||||||||||||||||||.||||||
NM_000207.3       51 CCTTCTGCCATGGCCCTGTGGATGCGCCTCCTGCCCCTGCTGGCGCTGCT    100

NM_001008996.    101 GGCCCTCTGGGGACCTGACCCAGCCTCGGCCTTTGTGAACCAACACCTGT    150
                     |||||||||||||||||||||||||.|.||||||||||||||||||||||
NM_000207.3      101 GGCCCTCTGGGGACCTGACCCAGCCGCAGCCTTTGTGAACCAACACCTGT    150

NM_001008996.    151 GCGGCTCCCACCTGGTGGAAGCTCTCTACCTAGTGTGCGGGGAACGAGGC    200
                     |||||||.||||||||||||||||||||||||||||||||||||||||||
NM_000207.3      151 GCGGCTCACACCTGGTGGAAGCTCTCTACCTAGTGTGCGGGGAACGAGGC    200
```

k-mer classification -> read classification

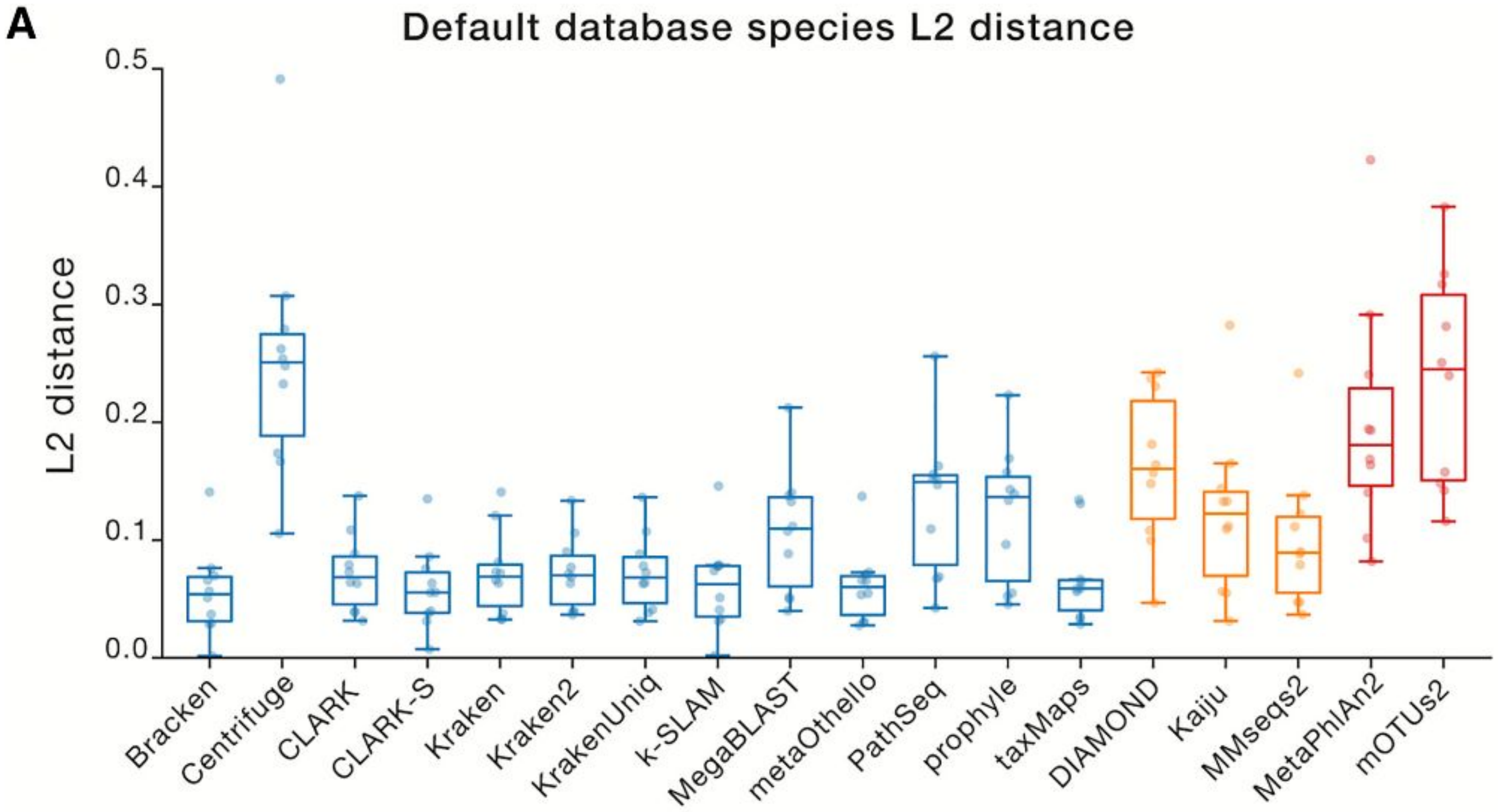

Query sequence (NGS read)
k-mers
K-mer to LCA mapping
(pre-computed database)
Taxonomy tree
Examine hit taxa
and ancestors
Classification
tree and path
1
10
0
4
1
Sequence classified as belonging to leaf of
classification (highest-weighted RTL) path

Evaluation

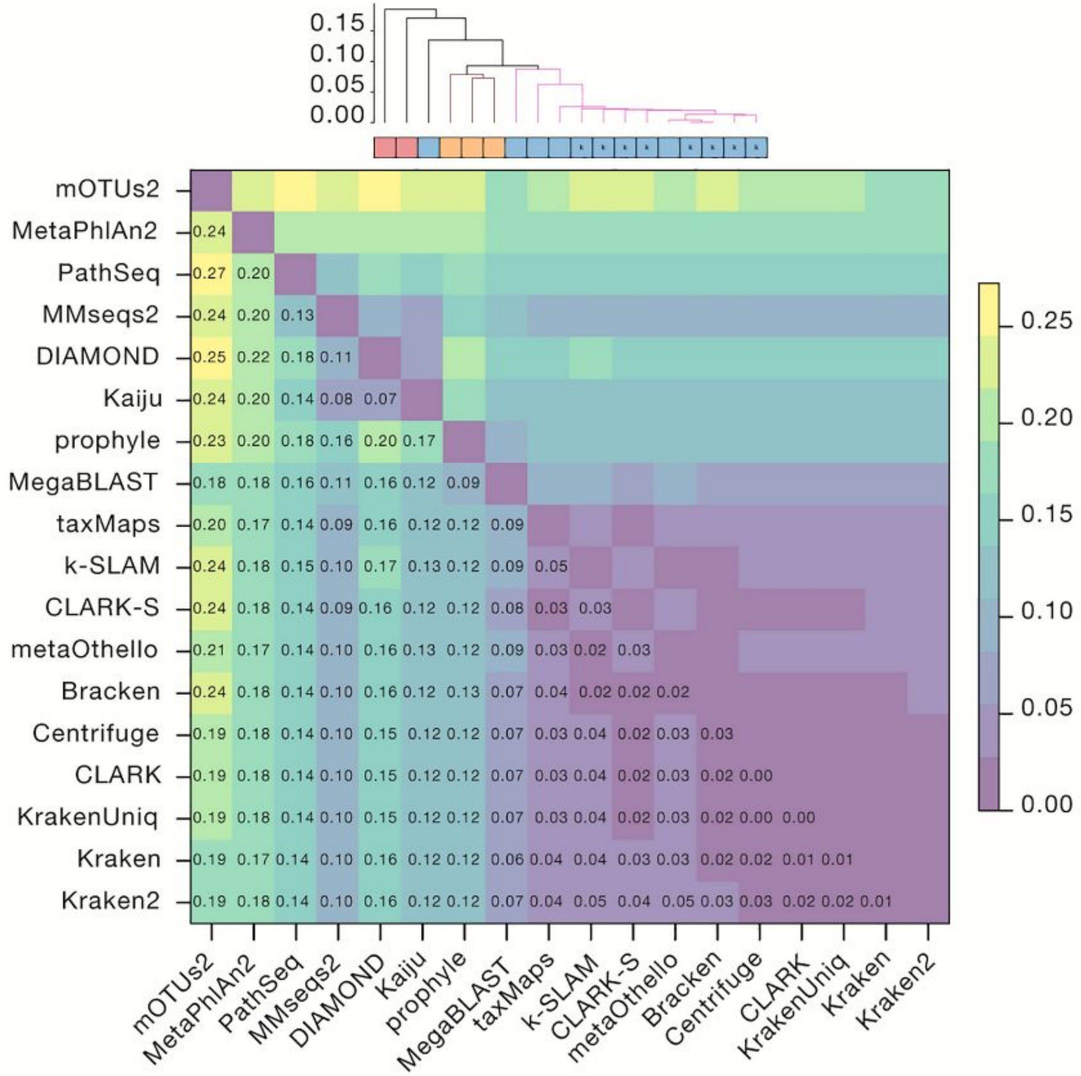


Type	Classifier	Memory Required	Time Required	Reference
DNA	CLARK-S	170 Gb	40 min	Ounit and Lonardi, 2016
	Kraken	190 Gb	1 min	Wood and Salzberg, 2014
	Kraken2	36 Gb	1 min	Wood and Salzberg, 2014
	KrakenUniq	200 Gb	1 min	Breitwieser et al., 2018
Protein	DIAMOND	110 Gb (varies)	10 min	Buchfink et al., 2015
	MMseqs2	85 Gb (varies)	9 h	Steinegger and Söding, 2017
Markers	MetaPhlAn2	2 Gb	1 min	Truong et al., 2015

Benchmarking Metagenomics Tools for Taxonomic Classification, Ye et al. 2019
https://doi.org/10.1016/j.cell.2019.07.010


A
Default database species L2 distance
L2 distance
0.0
0.1
0.2
0.3
0.4
0.5
Bracken
Centrifuge
CLARK
CLARK-S
Kraken
Kraken2
KrakenUniq
k-SLAM
MegaBLAST
metaOthello
PathSeq
prophyle
taxMaps
DIAMOND
Kaiju
MMseqs2
MetaPhlAn2
mOTUs2

High similarity between Kraken 1 and Kraken 2



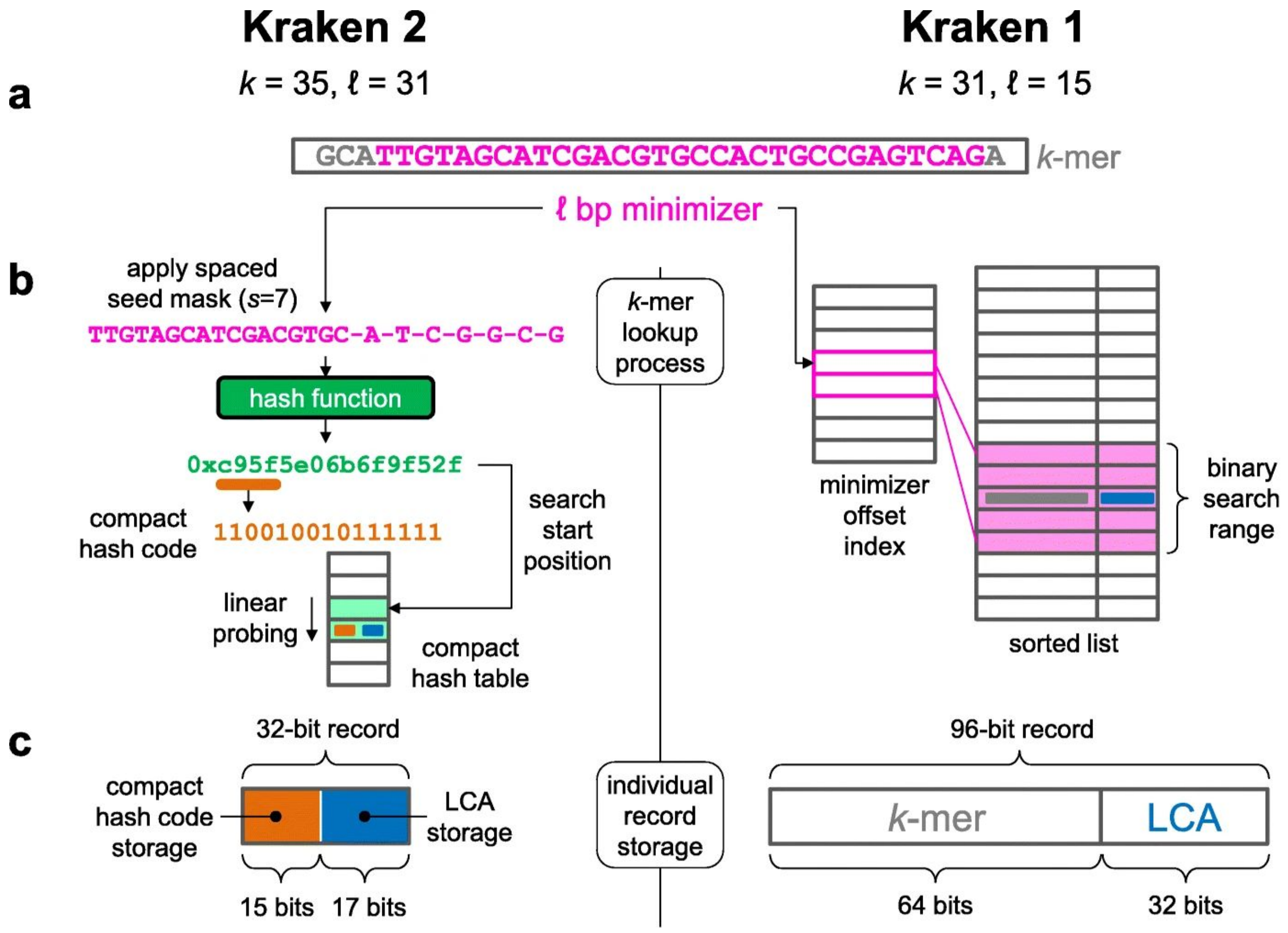
	mOTUs2	MetaPhlAn2	PathSeq	MMseqs2	DIAMOND	Kaiju	prophyle	MegaBLAST	taxMaps	k-SLAM	CLARK-S	metaOthello	Bracken	Centrifuge	CLARK	KrakenUniq	Kraken	Kraken2
mOTUs2																		
MetaPhlAn2	0.24																	
PathSeq	0.27	0.20																
MMseqs2	0.24	0.20	0.13															
DIAMOND	0.25	0.22	0.18	0.11														
Kaiju	0.24	0.20	0.14	0.08	0.07													
prophyle	0.23	0.20	0.18	0.16	0.20	0.17												
MegaBLAST	0.18	0.18	0.16	0.11	0.16	0.12	0.09											
taxMaps	0.20	0.17	0.14	0.09	0.16	0.12	0.12	0.09										
k-SLAM	0.24	0.18	0.15	0.10	0.17	0.13	0.12	0.09	0.05									
CLARK-S	0.24	0.18	0.14	0.09	0.16	0.12	0.12	0.08	0.03	0.03								
metaOthello	0.21	0.17	0.14	0.10	0.16	0.13	0.12	0.09	0.03	0.02	0.03							
Bracken	0.24	0.18	0.14	0.10	0.16	0.12	0.13	0.07	0.04	0.02	0.02	0.02						
Centrifuge	0.19	0.18	0.14	0.10	0.15	0.12	0.12	0.07	0.03	0.04	0.02	0.03	0.03					
CLARK	0.19	0.18	0.14	0.10	0.15	0.12	0.12	0.07	0.03	0.04	0.02	0.03	0.02	0.00				
KrakenUniq	0.19	0.18	0.14	0.10	0.15	0.12	0.12	0.07	0.03	0.04	0.02	0.03	0.02	0.00	0.00			
Kraken	0.19	0.17	0.14	0.10	0.16	0.12	0.12	0.06	0.04	0.04	0.03	0.03	0.02	0.02	0.01	0.01		
Kraken2	0.19	0.18	0.14	0.10	0.16	0.12	0.12	0.07	0.04	0.05	0.04	0.05	0.03	0.03	0.02	0.02	0.01	

System Requirements and Colab

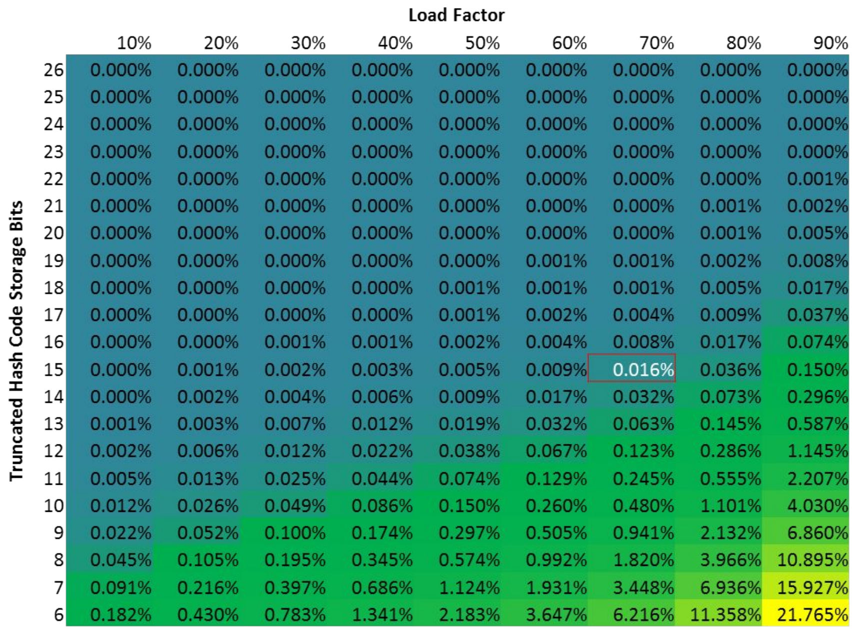
- 100 GB of disk space
- 29 GB of RAM
- “strongly suggest against using NFS storage”
- Use viral genomes instead

https://colab.research.google.com/drive/1DHLIjov7hSTg1fliDOCErUvl2dW-EM6X?usp=sharing

~End~

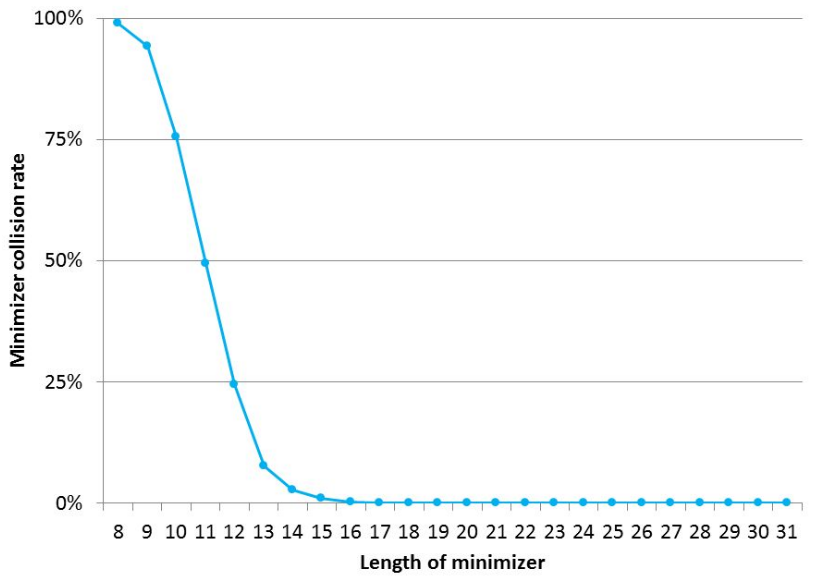

Kraken 2
Kraken 1
a
k = 35, ℓ = 31
k = 31, ℓ = 15
GCATTGTAGCATCGACGTGCCACTGCCGAGTCAGA
k-mer
ℓ bp minimizer
b
apply spaced seed mask (s=7)
TTGTAGCATCGACGTGC-A-T-C-G-G-C-G
hash function
0xc95f5e06b6f9f52f
compact hash code
110010010111111
search start position
linear probing
compact hash table
k-mer lookup process
minimizer offset index
binary search range
sorted list
c
32-bit record
compact hash code storage
LCA storage
15 bits
17 bits
individual record storage
96-bit record
k-mer
LCA
64 bits
32 bits

Hash Collisions? Not a Problem



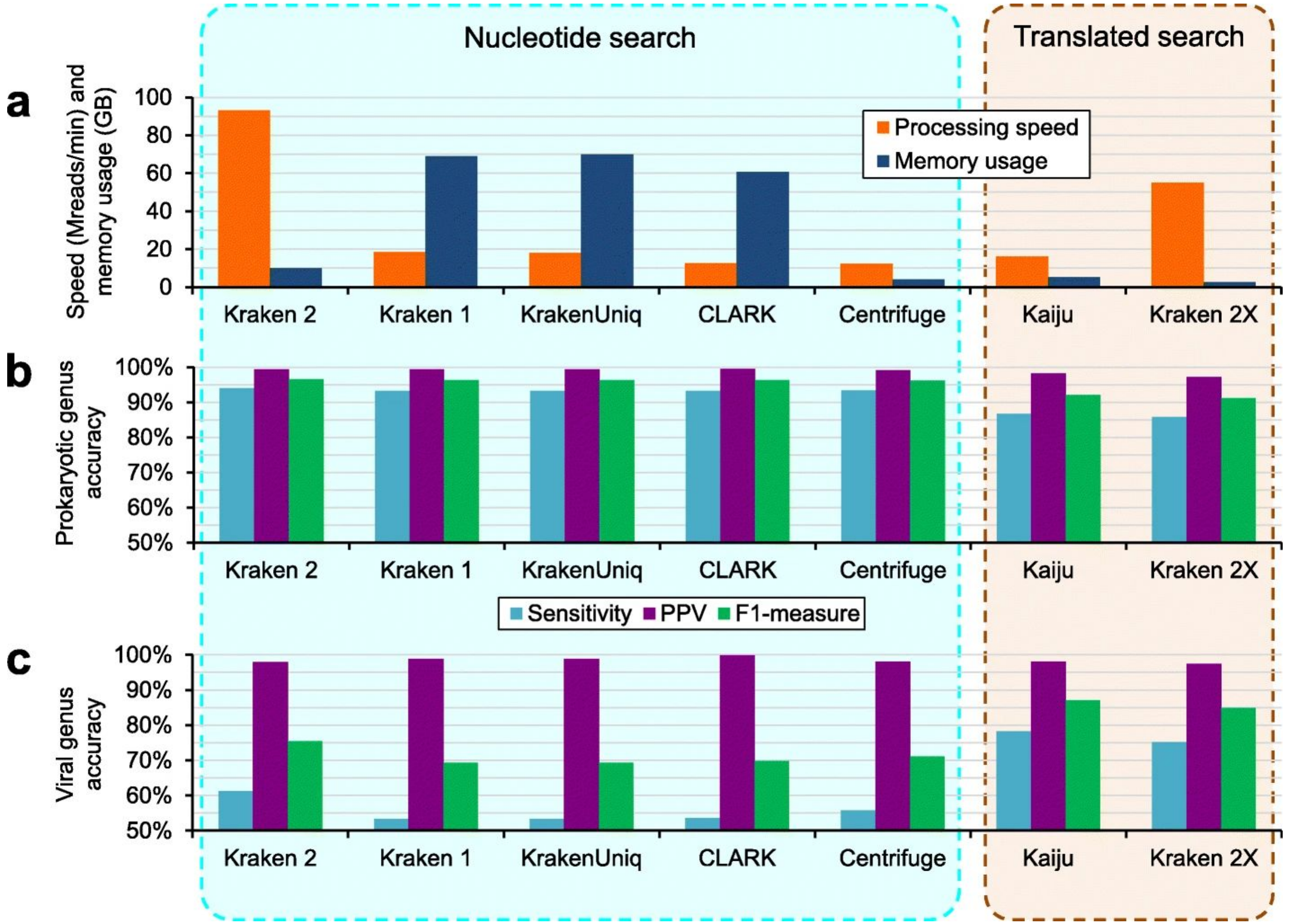
Load Factor / **Truncated Hash Code Storage Bits**

	10%	20%	30%	40%	50%	60%	70%	80%	90%
26	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%
25	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%
24	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%
23	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%
22	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.001%
21	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.001%	0.002%
20	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.001%	0.005%
19	0.000%	0.000%	0.000%	0.000%	0.000%	0.001%	0.001%	0.002%	0.008%
18	0.000%	0.000%	0.000%	0.000%	0.001%	0.001%	0.001%	0.005%	0.017%
17	0.000%	0.000%	0.000%	0.000%	0.001%	0.002%	0.004%	0.009%	0.037%
16	0.000%	0.000%	0.001%	0.001%	0.002%	0.004%	0.008%	0.017%	0.074%
15	0.000%	0.001%	0.002%	0.003%	0.005%	0.009%	0.016%	0.036%	0.150%
14	0.000%	0.002%	0.004%	0.006%	0.009%	0.017%	0.032%	0.073%	0.296%
13	0.001%	0.003%	0.007%	0.012%	0.019%	0.032%	0.063%	0.145%	0.587%
12	0.002%	0.006%	0.012%	0.022%	0.038%	0.067%	0.123%	0.286%	1.145%
11	0.005%	0.013%	0.025%	0.044%	0.074%	0.129%	0.245%	0.555%	2.207%
10	0.012%	0.026%	0.049%	0.086%	0.150%	0.260%	0.480%	1.101%	4.030%
9	0.022%	0.052%	0.100%	0.174%	0.297%	0.505%	0.941%	2.132%	6.860%
8	0.045%	0.105%	0.195%	0.345%	0.574%	0.992%	1.820%	3.966%	10.895%
7	0.091%	0.216%	0.397%	0.686%	1.124%	1.931%	3.448%	6.936%	15.927%
6	0.182%	0.430%	0.783%	1.341%	2.183%	3.647%	6.216%	11.358%	21.765%



Why do we even need that?

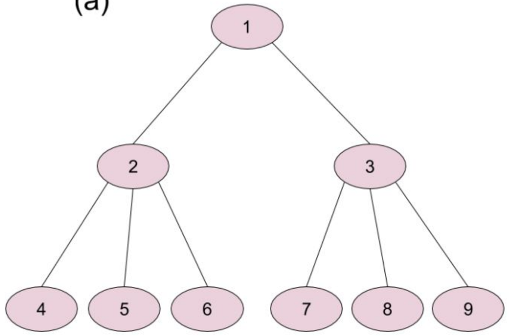
- Antibiotics can harm stem cell transplantation recipients
  https://doi.org/10.1126/scitranslmed.aaf2311


Nucleotide search
Translated search
a
Speed (Mreads/min) and memory usage (GB)
100
80
60
40
20
0
Processing speed
Memory usage
Kraken 2
Kraken 1
KrakenUniq
CLARK
Centrifuge
Kaiju
Kraken 2X
b
Prokaryotic genus accuracy
100%
90%
80%
70%
60%
50%
Sensitivity
PPV
F1-measure
c
Viral genus accuracy

Taxonomy

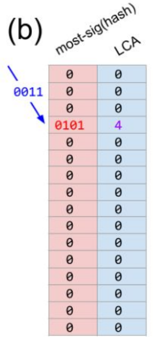
Insertions

(a)



Simple taxonomy with tax Ids

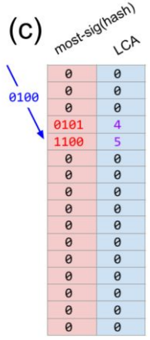
(b)



most-sig(hash)	LCA
0	0
0	0
0	0
0101	4
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0

Insert 8-bit minimizer hash with 0011 as least-sig bits, 0101 as most-sig (01010011), and 4 as taxon of origin. Immediately find empty slot.

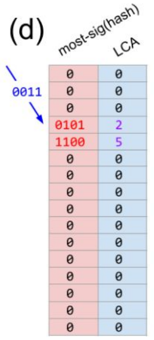
(c)



most-sig(hash)	LCA
0	0
0	0
0	0
0101	4
1100	5
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0

Insert 11000100, with 5 as taxon of origin. Immediately find empty slot.

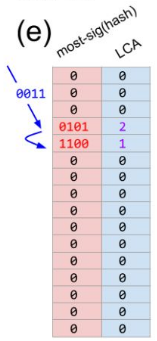
(d)



most-sig(hash)	LCA
0	0
0	0
0	0
0101	2
1100	5
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0

Insert 01010011, with 6 as taxon of origin. Immediately find slot with matching most-sig bits, set value = LCA(4, 6) = 2.

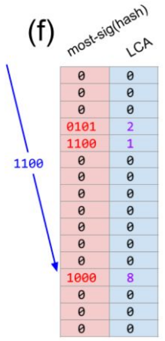
(e)



most-sig(hash)	LCA
0	0
0	0
0	0
0101	2
1100	1
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0

Insert 11000011, with 7 as taxon of origin. Scan forward 1 row before finding match for 1100. Note: **this is a hash table collision**. Set value = LCA(5, 7) = 1.

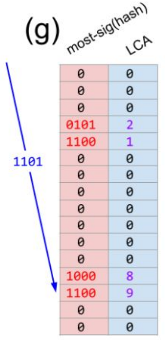
(f)



most-sig(hash)	LCA
0	0
0	0
0	0
0101	2
1100	1
0	0
0	0
0	0
0	0
0	0
0	0
0	0
1000	8
0	0
0	0
0	0

Insert 10001100, with 8 as taxon of origin. Immediately find empty slot.

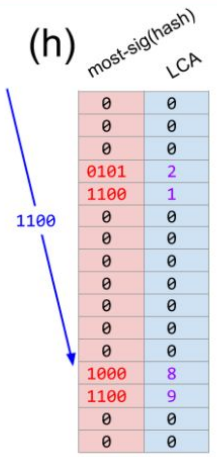
(g)



most-sig(hash)	LCA
0	0
0	0
0	0
0101	2
1100	1
0	0
0	0
0	0
0	0
0	0
0	0
0	0
1000	8
1100	9
0	0
0	0

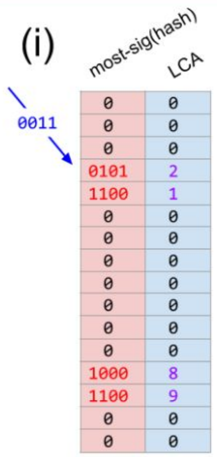
Insert 11001101, with 9 as taxon of origin. Immediately find empty slot.

Queries



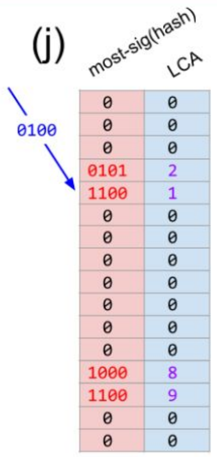
most-sig(hash)	LCA
0	0
0	0
0	0
0101	2
1100	1
0	0
0	0
0	0
0	0
0	0
0	0
0	0
1000	8
1100	9
0	0
0	0

Query 8-bit minimizer hash 10001100. Find 8, the correct answer for that hash.



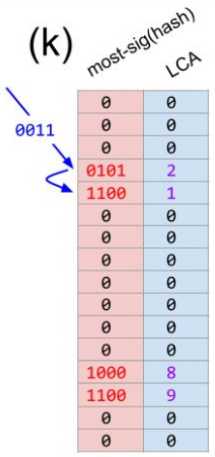
most-sig(hash)	LCA
0	0
0	0
0	0
0101	2
1100	1
0	0
0	0
0	0
0	0
0	0
0	0
0	0
1000	8
1100	9
0	0
0	0

Query 01010011. Find 2, the LCA of the taxa for the two identical hashes we inserted in steps b and d.



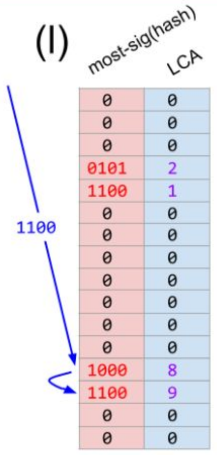
most-sig(hash)	LCA
0	0
0	0
0	0
0101	2
1100	1
0	0
0	0
0	0
0	0
0	0
0	0
0	0
1000	8
1100	9
0	0
0	0

Query 11000100. Find 1, the LCA of the taxa for two hashes, one matching the query and one (11000011) not. These were added in steps c and e.



most-sig(hash)	LCA
0	0
0	0
0	0
0101	2
1100	1
0	0
0	0
0	0
0	0
0	0
0	0
0	0
1000	8
1100	9
0	0
0	0

Query 11000011. Scan forward 1 row before finding matching least-sig bits. Find 1, the LCA of the taxa for two hashes, one matching the query and one (11000100) not. These were added in steps c and e.



most-sig(hash)	LCA
0	0
0	0
0	0
0101	2
1100	1
0	0
0	0
0	0
0	0
0	0
0	0
0	0
1000	8
1100	9
0	0
0	0

Query 11001100. Scan forward 1 row before finding matching least-sig bits. This yields the LCA 9, **which is spurious because this query hash was never inserted**.