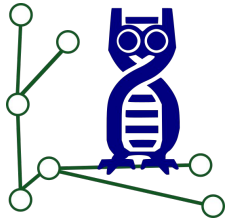
UNIVERSITÄT
DES
SAARLANDES



Fast gapped *k*-mer counting with subdivided multi-way bucketed Cuckoo hash tables

Sven Rahmann, Jens Zentgraf
Algorithmic Bioinformatics, Saarland University

DSB, 14.06.2022

gitlab.com/rahmannlab/hackgap

k-mer Counting

- Given
  - ➢ FASTA or FASTQ input files with DNA sequences,
  - ➢ integer *k* (or a gapped *k*-mer mask),

- compute
  - ➢ a key-value store that stores the count of each *k*-mer in the collection,
  - ➢ ideally with very fast subsequent lookup time.

k-mer counting tools

- **kmc3** and **gerbil**
  - ➢ fast, disk based
  - ➢ specialized on *k*-mer counting
  - ➢ fast because of minimizers and super-*k*-mers
  - ➢ hard to adapt to gapped *k*-mers
    (because of minimizers would change from *k*-mer to *k*-mer)

- **hackgap** (our tool)
  - ➢ in-memory only
  - ➢ uses a Cuckoo hash table
  - ➢ no use of minimizers or similar concepts
  - ➢ contiguous and gapped *k*-mers treated (more or less) in the same way
  - ➢ implemented with Python and numba (just-in-time compiler)

Contiguous *k*-mers

- Easy to handle
- Prone to errors
  - ➢ one error changes *k* **consecutive** *k*-mers
- 3 errors to change all *k*-mers

```
###
TACAGATATA
TAC GAT
 ACA ATA
  CAG TAT
   AGA ATA
```

Gapped *k*-mers

- Window size *w* (*w* = 7)
- Significant positions *k* (*k* = 3)
- Mask or tuple of offsets
  - `Mask: #__#__#`
  - `Tuple: (0,3,6)`
- symmetric masks only
- More complex to handle
- **error tolerant**
  - one error changes *k* kmers
  - affected *k*-mers are not consecutive
- empirically: more unique *k*-mers

```
#__#__#
TACAGATATA
T__A__T
 A__G__A
  C__A__T
   A__T__A
```

Gapped *k*-mers

- Window size *w* (*w* = 7)
- Significant positions *k* (*k* = 3)
- Mask or tuple of offsets
  - `Mask: #__#__#`
  - `Tuple: (0,3,6)`
- symmetric masks only
- More complex to handle
- **error tolerant**
  - one error changes *k* kmers
  - affected *k*-mers are not consecutive
- empirically: more unique *k*-mers

	mask (31,25)	max d_H	intact *k*-mers	covered base pairs
k25	###########################	3	1	25
m2	####_####_###_###_###_####_####	4	2	32
m3	####_###_###_#####_###_###_####	4	2	32
m4	###_##_#####_#####_#####_##_###	4	3	34

```
#__#__#
TACAGATATA
T__A__T
 A__G__A
  C__A__T
   A__T__A
```

DNA bit encoding and canonical codes (max)

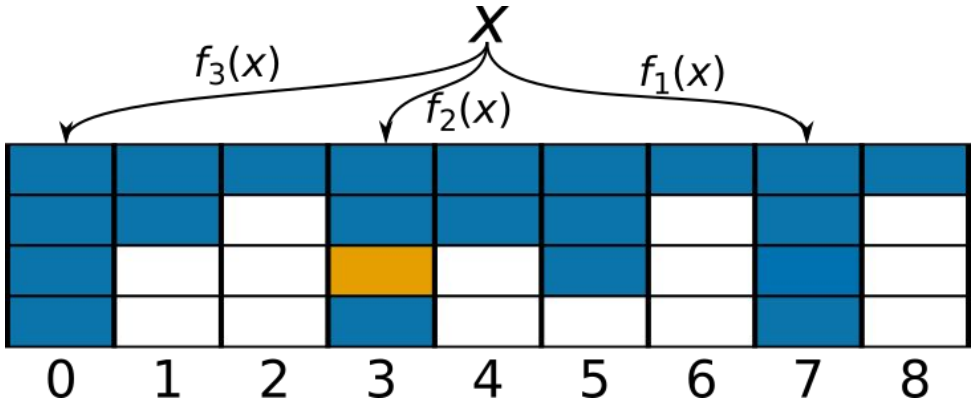
- 2 bit encoding
  - A→$(00)_2$, C→$(01)_2$,
  - G→$(10)_2$, T→$(11)_2$
- *k*-mer integer encoding
  - base-4 number c = enc(*k*-mer)
  - c = enc(TAC) = $(110001)_2$ = 49
- reverse complement (rc)
  - A↔T, C↔G
  - TAC ↔ GTA
  - enc(GTA) = $(101100)_2$ = 44
- Canonical code (max)
  - Maximum of k-mer and reverse complement
  - *max*{enc(TAC), enc(GTA)}
  - = *max*{49, 44} = 49
- Better encoding for odd *k* (saving 1 bit) discussed at DSB yesterday

```
###
TACAGATATA
TAC GAT
 ACA ATA
  CAG TAT
   AGA ATA
```

3-way Cuckoo hashing with buckets of size 4

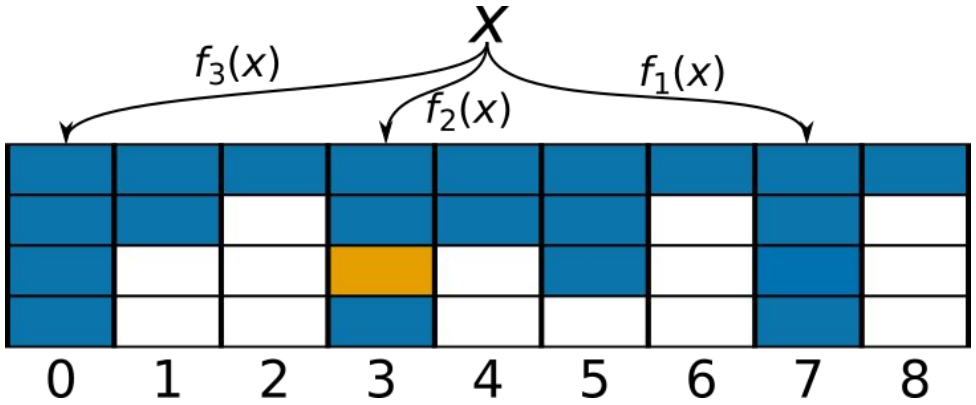
- 3 hash functions:
- Each maps a *k*-mer (***X***) to a bucket.
- Each bucket can store up to 4 elements.
- Idea: bucket fits within a cache line.

- 12 possible locations for each element.
- At worst 3 memory lookups (cache misses), often only 1 or 2.



Insertion by random walk

- Insert *x*:
  try buckets $f_1(x)$, $f_2(x)$, $f_3(x)$ in order;
  insert into first bucket with space available.
- If all full, evict a random element, place current element into now free slot.
- Re-insert evicted element into different slot.
- May cause another eviction…
  ⇒ random walk through table.
- Limit length of walk (e.g. 500 steps). Fail if limit reached.



Quotienting

Keys are encoded canonical *k*-mers (half of set $[4^k] := \{0, .., 4^k\text{-}1\}$).

- **Step 1:** Bijective randomizing function $[4^k] \to [4^k]$ with *a* odd

$$g_{a,b}(x) := [a \cdot (\mathrm{rot}_k(x) \text{ xor } b)] \bmod 4^k$$

- **Step 2:** Map to buckets (simply mod *p*: number of buckets). Define

$$f(x) := g_{a,b}(x) \bmod p \quad \text{and} \quad q(x) := g_{a,b}(x) \,//\, p\ .$$

- Then *x* can be uniquely reconstructed
  from *f*(*x*) ("hash value, "bucket number") and *q*(*x*) ("fingerprint", "quotient").
  Sufficient to store *q*(*x*) in bucket *f*(*x*) (and which hash function was chosen).

Parallelization

Lock free approach using CAS

- Multiple threads working in one hashtable
- CAS is atomic → no locks
- Not compatible with bit packing

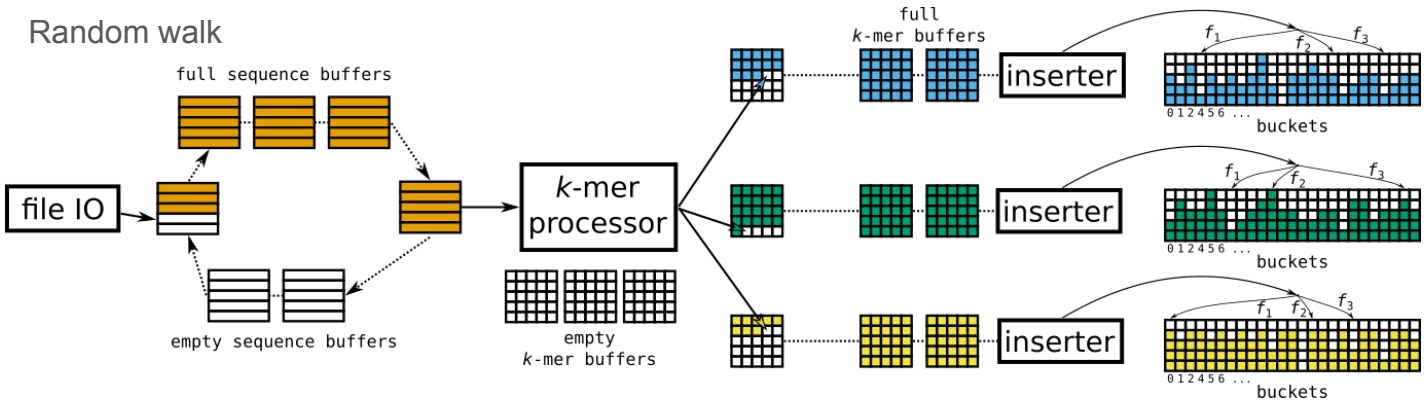
Subtables

- Use multiple (*h*,*b*) Cuckoo hash tables
- One thread per table
  - ➢ No locks needed
- No changes in the hash table
- Easy to scale
- Producer-consumer strategy

Insertion with subtables

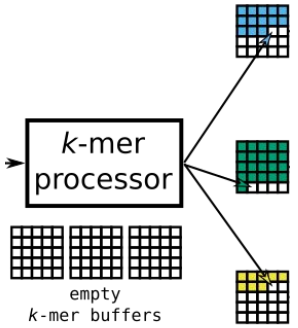
- Producer-consumer-strategy
- File IO in Python
- Producer: ***k*-mer Processor**
  - ➢ Calculate integer representation
  - ➢ Calculate *k*-mers of reads
  - ➢ Distribute *k*-mers to subtable
- Consumer: **inserter**
  - ➢ One inserter per subtable
  - ➢ Insert all *k*-mers in a subtable
  - ➢ Random walk

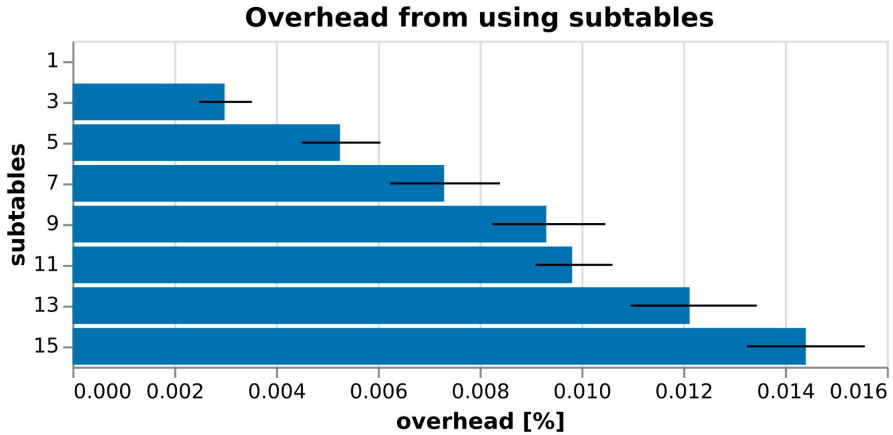
- Buffered communication
  - ➢ Sequence buffers (file IO → k-mer processor): Read one block from a file (~8MB)
  - ➢ *k*-mer buffers:
    - Store all *k*-mers
    - Multiple buffers per subtable



Subtable distribution

- Linear hash function (soon maybe a new encoding for *cc*)
- *k*-mers are not perfectly evenly distributed, but **almost perfectly**.
- Double benefit from quotienting



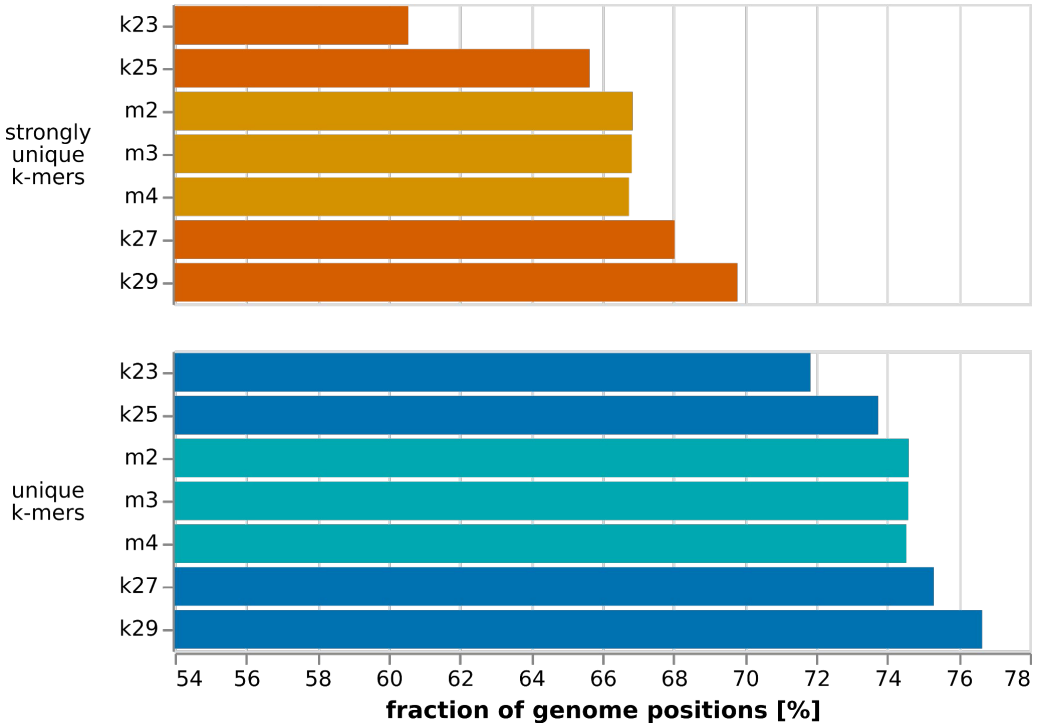


Strongly unique *k*-mers in t2t

shape (*w*,*k*) = (31,25)

m2	####_####_###_###_###_####_####
m3	####_###_###_#####_###_###_####
m4	###_##_#####_#####_#####_##_###

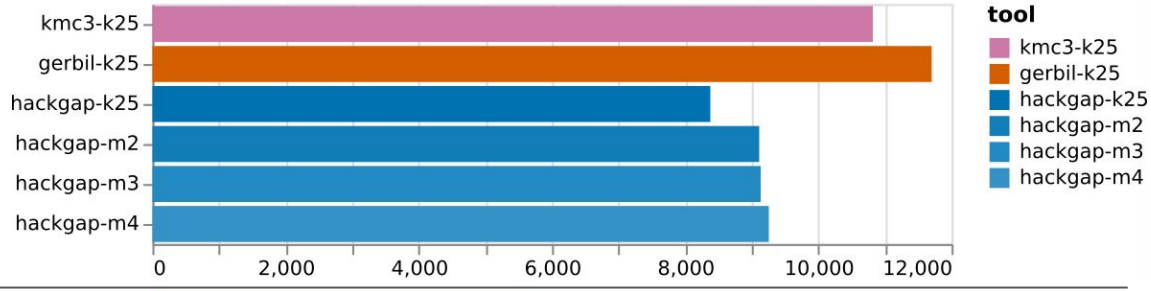
- strongly unique: unique and there is no other *k*-mer with Hamming distance 1



Göttingen minipigs and t2t Human genome reference

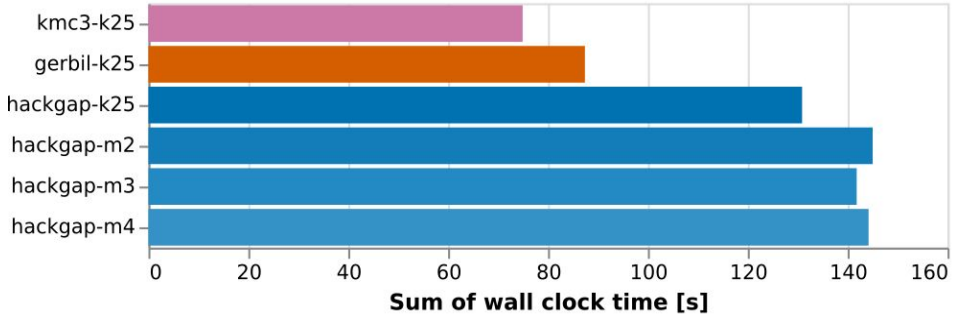
Göttingen minipig:

- 10 samples (PE fastq)
- 36.788 - 42.540 Gbp
- 16GB Memory
- 24 threads / 5 subtables



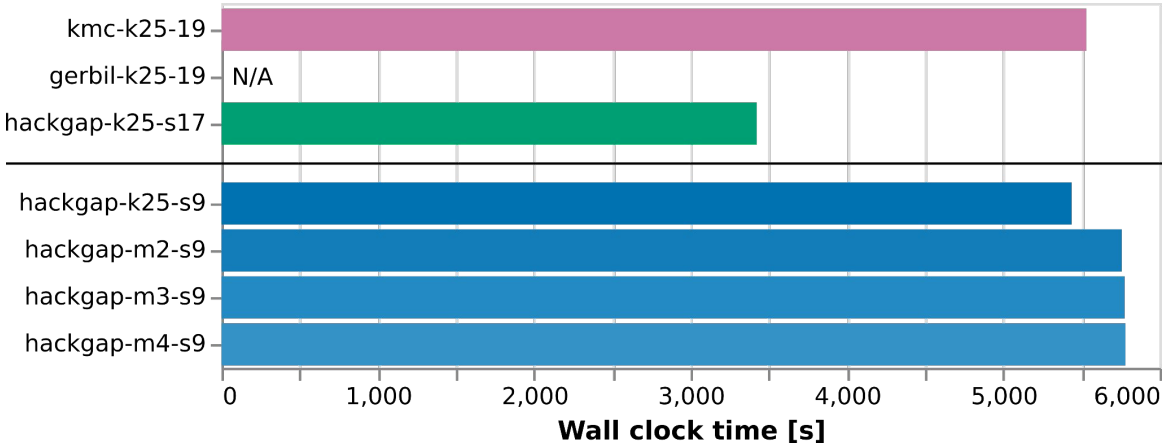
t2t reference:

- 1 fasta file
- 3.117 Gbp
- 16GB Memory
- 16 threads / 5 subtables



GIAB Ashkenazim trio

- all counted together
- 314.554 Gbp



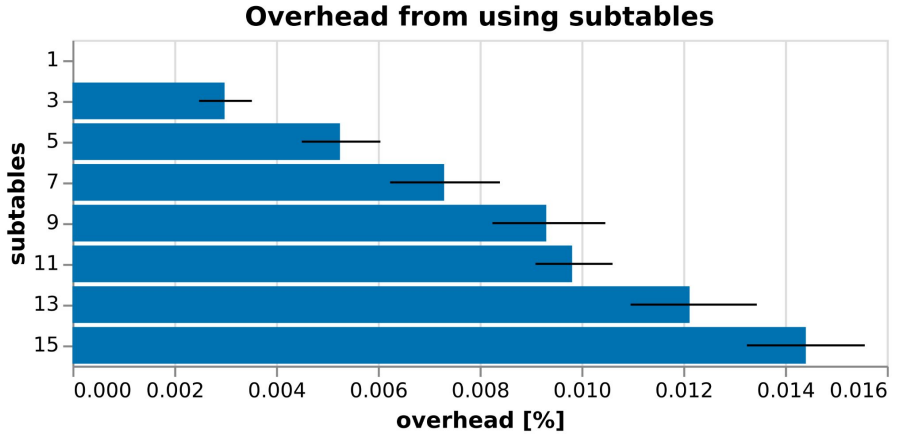
Limitations and advantages

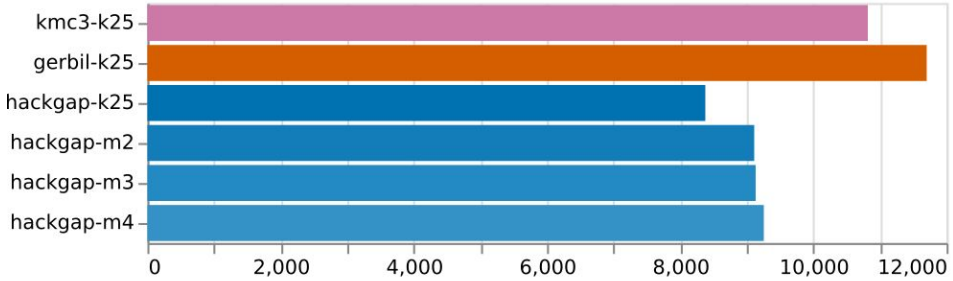
- In-memory
  (Limited by memory size)
- Fixed size
  (must be known in advance)
- $k \leq 32$ (64 bit integer encoding)
- File IO in pure Python

- No temporary files
- Supports gapped *k*-mers
- Does not use minimizers
- Only slight increase in runtime with gapped k-mers

Hackgap - Summary

- Subdivided hash tables
- Support gapped *k*-mers with a small increase in runtime
- Fast lookup times
- Can compete with **kmc3** and **gerbil**





gitlab.com/rahmannlab/hackgap