The Jaccard Index: general concept

- this section is based on[1] referred to by Ondov et al. in the following
- several frames are based on the presentation of the topic by Leo Förster, Genome Informatics Seminar, Wintersemester 2016/2017
- overall goal: compute a distance between two sequences
- distance can be determined by looking at what is common
- the more in common, the smaller the distance
- naive approach via Jaccard Index[2]

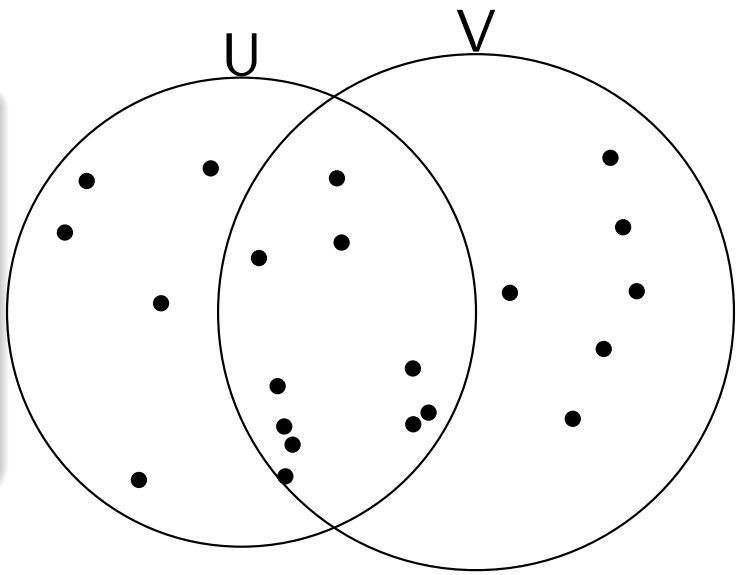
[1] Ondov, B. D., Treangen, T. J., Melsted, P., Mallonee, A. B., Bergman, N. H., Koren, S., and Phillippy, A. M. (2016).Mash: fast genome and metagenome distance estimation using MinHash. *Genome Biology*, 17(1):1–14.

[2] Jaccard, P (1901), Etude comparative de la distribution florale dans une portion des Alpes et des Jura, *Bullet. de la Société Vaudoise des Sciences Natur., 37: 547-579*

The Jaccard Index: general concept

Jaccard Index

- is fraction of shared elements of two sets: $J(U,V) = \frac{|U \cap V|}{|U \cup V|}$



$$J(U,V) = \frac{|U \cap V|}{|U \cup V|} = \frac{10}{21} = 0.4762$$

Properties

- $0 \leq |U \cap V| \leq |U \cup V| \Rightarrow 0 \leq J(U,V) \leq 1$
- $U \cap V = \emptyset \Rightarrow J(U,V) = \frac{|U \cap V|}{|U \cup V|} = \frac{|\emptyset|}{|U \cup V|} = \frac{0}{|U \cup V|} = 0$
- $U = V \Rightarrow J(U,V) = \frac{|U \cap V|}{|U \cup V|} = \frac{|U|}{|U|} = 1$

- the comparison of sequences using the Jaccard index is based on the sets $G_q(u)$ and $G_q(v)$ of all q-grams of u and v, respectively.
- that is, one computes $J(U, V)$ for $U = G_q(u)$ and $V = G_q(v)$
- so the black dots in previous figure represent q-grams
- abbreviation: $J(u, v) = J(G_q(u), G_q(v))$

Efficiency

- common q-grams can be efficiently counted:
  - for u and v enumerate and encode q-grams as integers (as done before)
  - store integer codes in two lists and sort each list using radix sort
  - find common elements in two sorted integer lists by a merging approach (see following slides)
- for $m = |u|$, $n = |v|$: $O(\underbrace{m+n}_{\text{enumerate}} + \underbrace{m+n}_{\text{sort}} + \underbrace{m+n}_{\text{merge}})$ time

$\Rightarrow$ $O(m+n)$ space and time (optimal)

Algorithm 1 (Count common elements in two sorted lists of integers)

Input: *ulist*, *vlist*: sorted lists of integers and without duplicates
Output: number of common elements

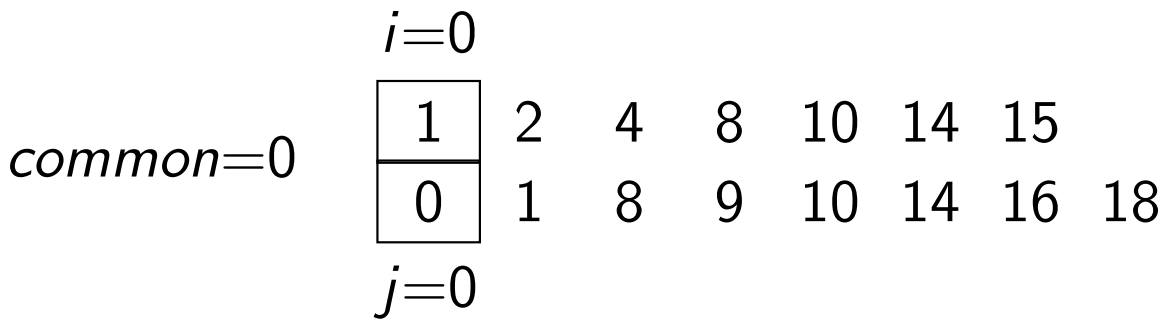
1: $(i, j, common) \leftarrow (0, 0, 0)$
2: **while** $i < |ulist|$ and $j < |vlist|$ **do**
3: **if** $ulist[i] < vlist[j]$ **then**
4: $i \leftarrow i + 1$
5: **else**
6: **if** $ulist[i] > vlist[j]$ **then**
7: $j \leftarrow j + 1$
8: **else**
9: $(i, j, common) \leftarrow (i + 1, j + 1, common + 1)$
10: **end if**
11: **end if**
12: **end while**
13: **print** *common*

Count common elements in sorted lists of integers

$$ulist[i] < vlist[j] \;\Rightarrow\; i \leftarrow i + 1$$

$$ulist[i] > vlist[j] \;\Rightarrow\; j \leftarrow j + 1$$

$$ulist[i] = vlist[j] \;\Rightarrow\; (i, j, common) \leftarrow (i + 1, j + 1, common + 1)$$

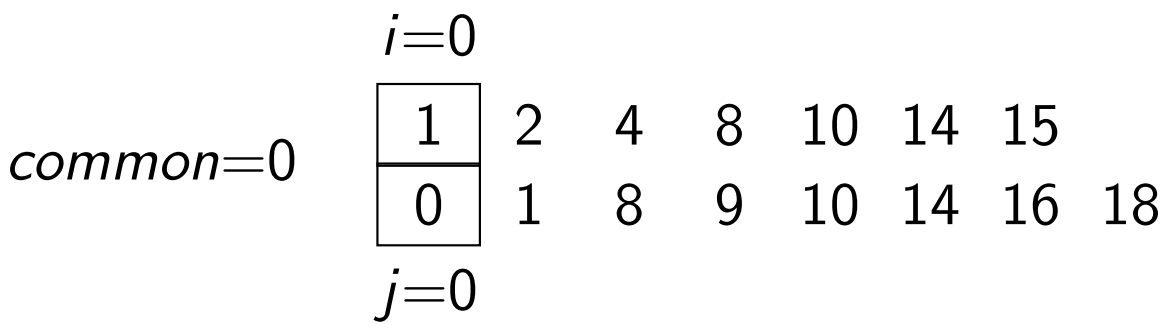


Count common elements in sorted lists of integers

$$ulist[i] < vlist[j] \;\Rightarrow\; i \leftarrow i+1$$

$$\boxed{ulist[i] > vlist[j] \;\Rightarrow\; j \leftarrow j+1}$$

$$ulist[i] = vlist[j] \;\Rightarrow\; (i, j, common) \leftarrow (i+1, j+1, common+1)$$

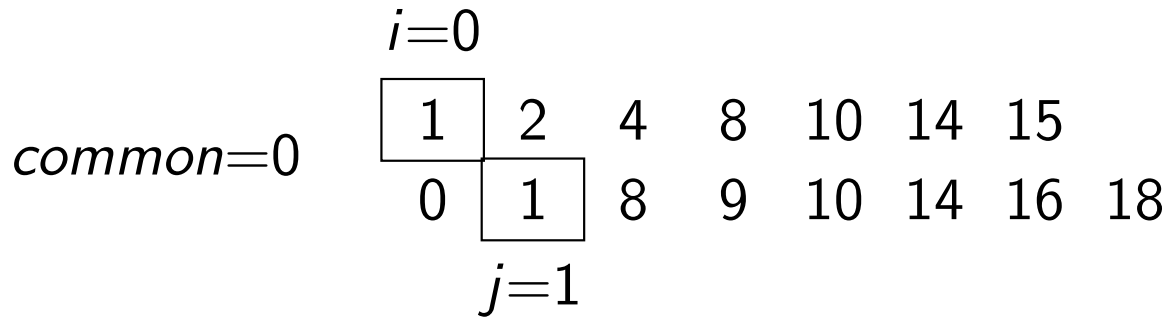


Count common elements in sorted lists of integers

$$ulist[i] < vlist[j] \;\Rightarrow\; i \leftarrow i + 1$$

$$ulist[i] > vlist[j] \;\Rightarrow\; j \leftarrow j + 1$$

$$ulist[i] = vlist[j] \;\Rightarrow\; (i, j, common) \leftarrow (i + 1, j + 1, common + 1)$$

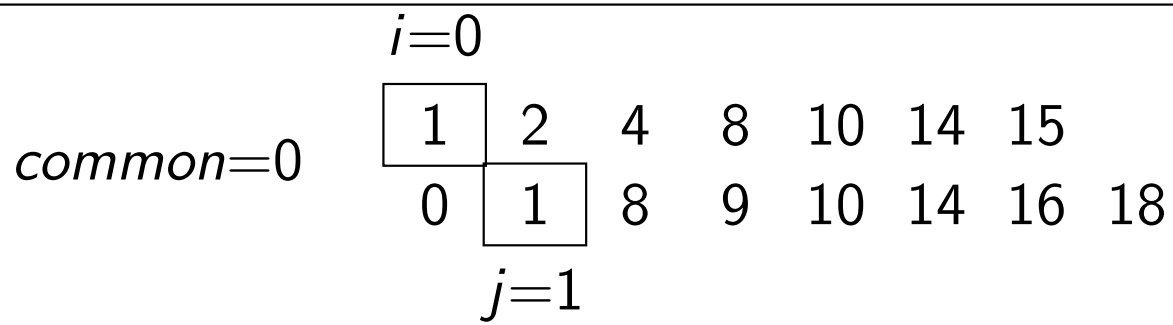


Count common elements in sorted lists of integers

$$ulist[i] < vlist[j] \;\Rightarrow\; i \leftarrow i + 1$$

$$ulist[i] > vlist[j] \;\Rightarrow\; j \leftarrow j + 1$$

$$\boxed{ulist[i] = vlist[j] \;\Rightarrow\; (i, j, common) \leftarrow (i + 1, j + 1, common + 1)}$$

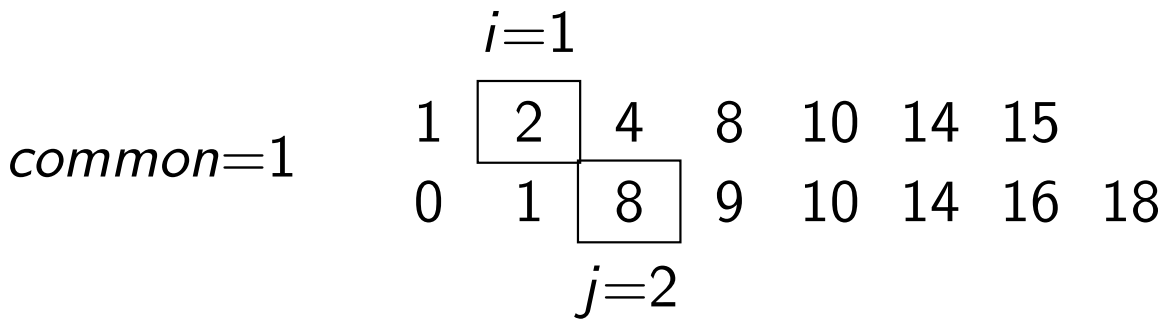


Count common elements in sorted lists of integers

$$ulist[i] < vlist[j] \;\Rightarrow\; i \leftarrow i + 1$$

$$ulist[i] > vlist[j] \;\Rightarrow\; j \leftarrow j + 1$$

$$ulist[i] = vlist[j] \;\Rightarrow\; (i, j, common) \leftarrow (i + 1, j + 1, common + 1)$$

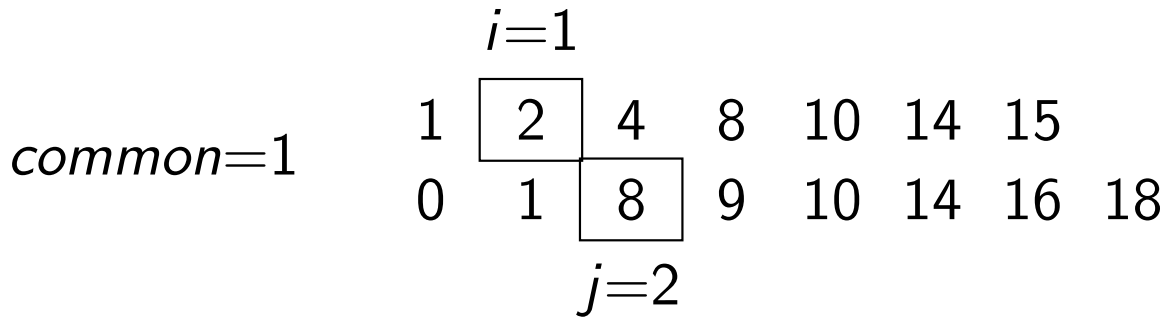


Count common elements in sorted lists of integers

$$\boxed{ulist[i] < vlist[j] \;\Rightarrow\; i \leftarrow i+1}$$

$$ulist[i] > vlist[j] \;\Rightarrow\; j \leftarrow j+1$$

$$ulist[i] = vlist[j] \;\Rightarrow\; (i, j, common) \leftarrow (i+1, j+1, common+1)$$

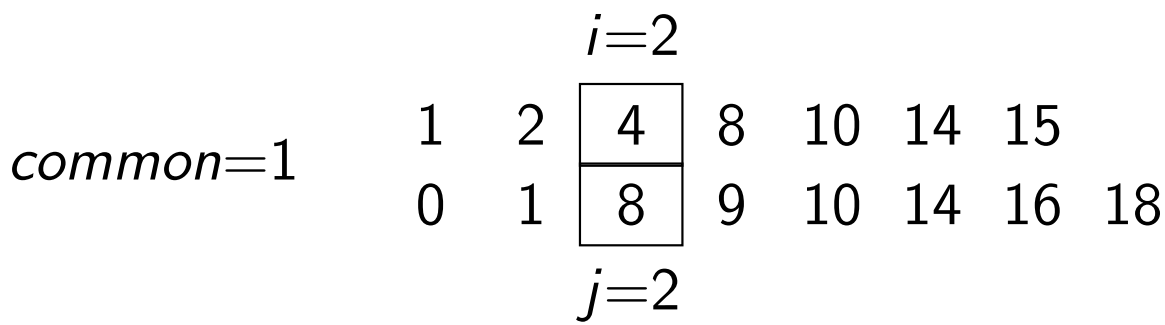


Count common elements in sorted lists of integers

$$ulist[i] < vlist[j] \;\Rightarrow\; i \leftarrow i + 1$$

$$ulist[i] > vlist[j] \;\Rightarrow\; j \leftarrow j + 1$$

$$ulist[i] = vlist[j] \;\Rightarrow\; (i, j, common) \leftarrow (i + 1, j + 1, common + 1)$$

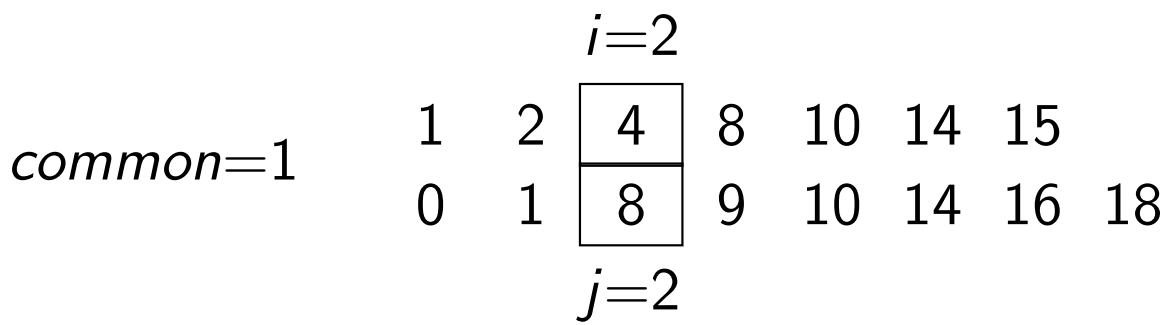


Count common elements in sorted lists of integers

$$\boxed{ulist[i] < vlist[j] \;\Rightarrow\; i \leftarrow i+1}$$

$$ulist[i] > vlist[j] \;\Rightarrow\; j \leftarrow j+1$$

$$ulist[i] = vlist[j] \;\Rightarrow\; (i, j, common) \leftarrow (i+1, j+1, common+1)$$

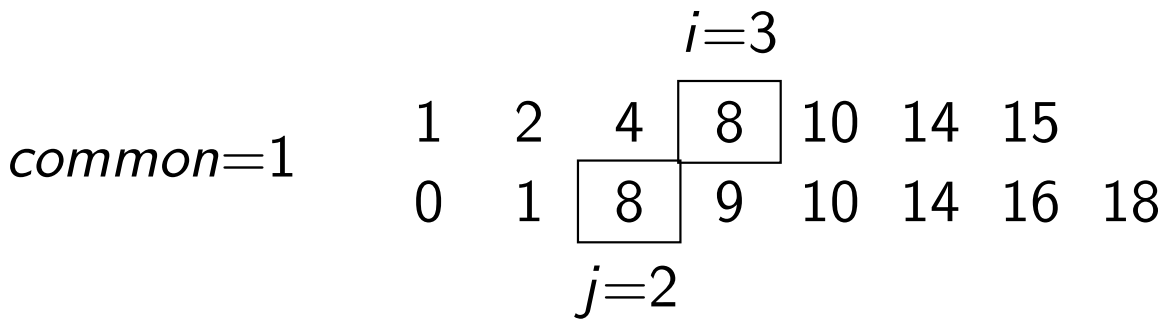


Count common elements in sorted lists of integers

$$ulist[i] < vlist[j] \;\Rightarrow\; i \leftarrow i + 1$$

$$ulist[i] > vlist[j] \;\Rightarrow\; j \leftarrow j + 1$$

$$ulist[i] = vlist[j] \;\Rightarrow\; (i, j, common) \leftarrow (i + 1, j + 1, common + 1)$$

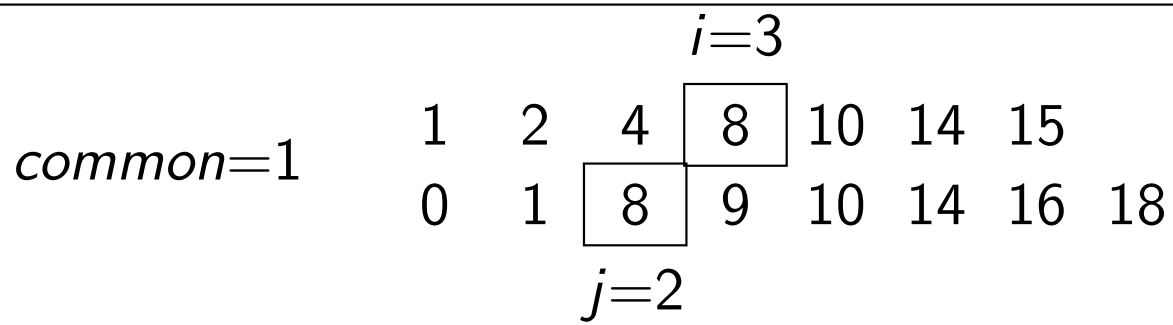


Count common elements in sorted lists of integers

$$ulist[i] < vlist[j] \;\Rightarrow\; i \leftarrow i + 1$$

$$ulist[i] > vlist[j] \;\Rightarrow\; j \leftarrow j + 1$$

$$\boxed{ulist[i] = vlist[j] \;\Rightarrow\; (i, j, common) \leftarrow (i + 1, j + 1, common + 1)}$$

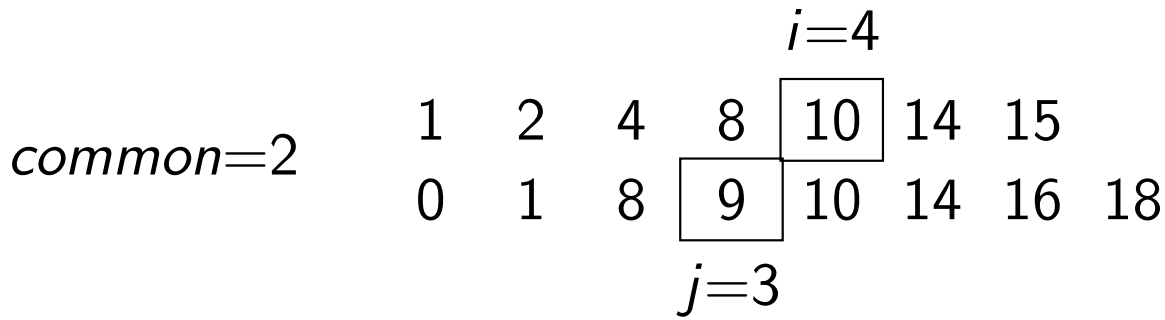


Count common elements in sorted lists of integers

$$ulist[i] < vlist[j] \;\Rightarrow\; i \leftarrow i + 1$$

$$ulist[i] > vlist[j] \;\Rightarrow\; j \leftarrow j + 1$$

$$ulist[i] = vlist[j] \;\Rightarrow\; (i, j, common) \leftarrow (i + 1, j + 1, common + 1)$$

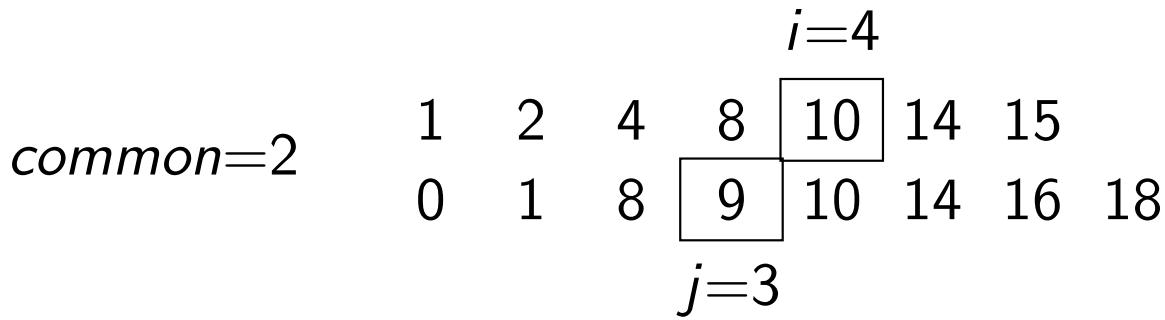


Count common elements in sorted lists of integers

$$ulist[i] < vlist[j] \;\Rightarrow\; i \leftarrow i+1$$

$$\boxed{ulist[i] > vlist[j] \;\Rightarrow\; j \leftarrow j+1}$$

$$ulist[i] = vlist[j] \;\Rightarrow\; (i, j, common) \leftarrow (i+1, j+1, common+1)$$

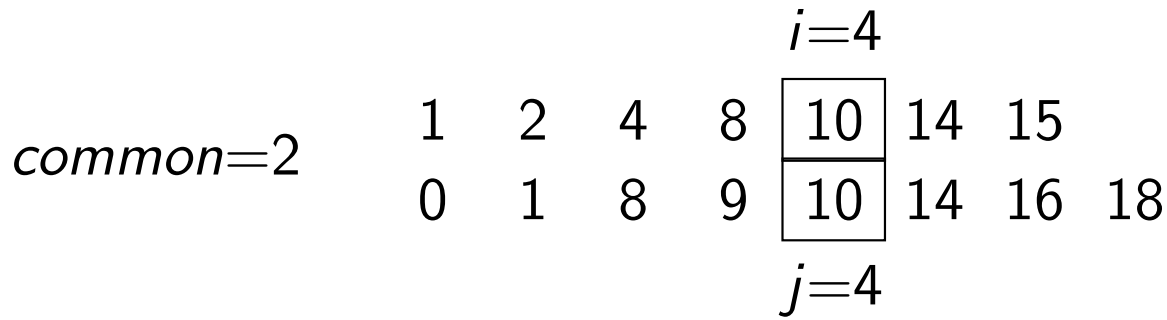


Count common elements in sorted lists of integers

$$ulist[i] < vlist[j] \;\Rightarrow\; i \leftarrow i + 1$$

$$ulist[i] > vlist[j] \;\Rightarrow\; j \leftarrow j + 1$$

$$ulist[i] = vlist[j] \;\Rightarrow\; (i, j, common) \leftarrow (i + 1, j + 1, common + 1)$$

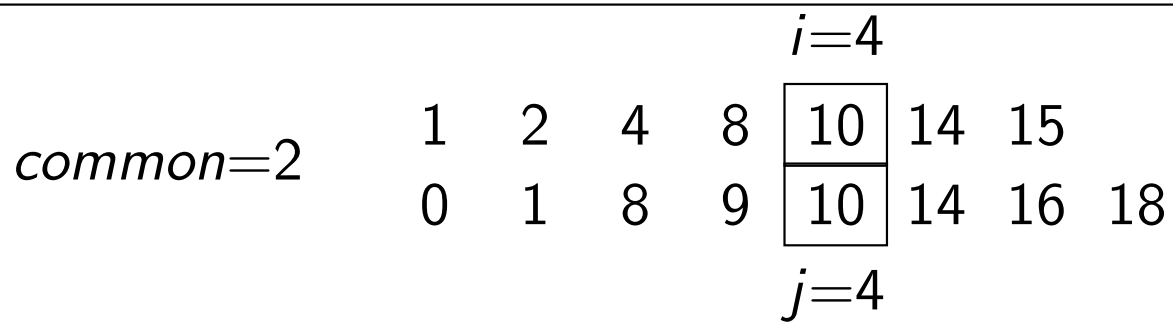


Count common elements in sorted lists of integers

$$ulist[i] < vlist[j] \;\Rightarrow\; i \leftarrow i + 1$$

$$ulist[i] > vlist[j] \;\Rightarrow\; j \leftarrow j + 1$$

$$\boxed{ulist[i] = vlist[j] \;\Rightarrow\; (i, j, common) \leftarrow (i + 1, j + 1, common + 1)}$$

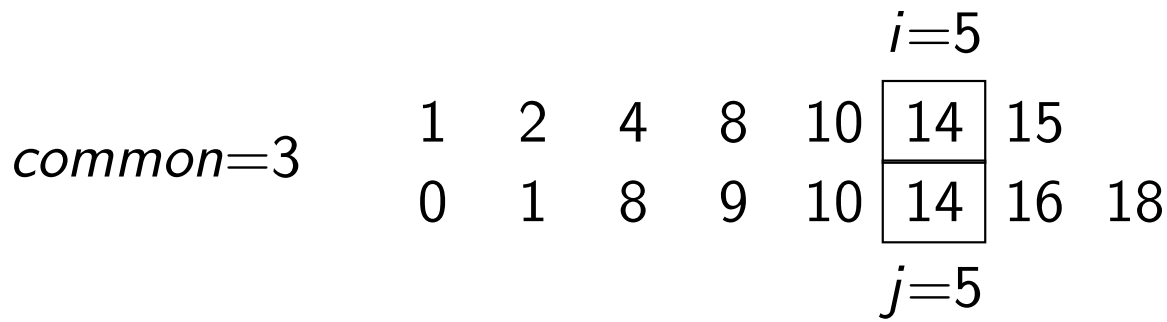


Count common elements in sorted lists of integers

$$ulist[i] < vlist[j] \;\Rightarrow\; i \leftarrow i + 1$$

$$ulist[i] > vlist[j] \;\Rightarrow\; j \leftarrow j + 1$$

$$ulist[i] = vlist[j] \;\Rightarrow\; (i, j, common) \leftarrow (i + 1, j + 1, common + 1)$$

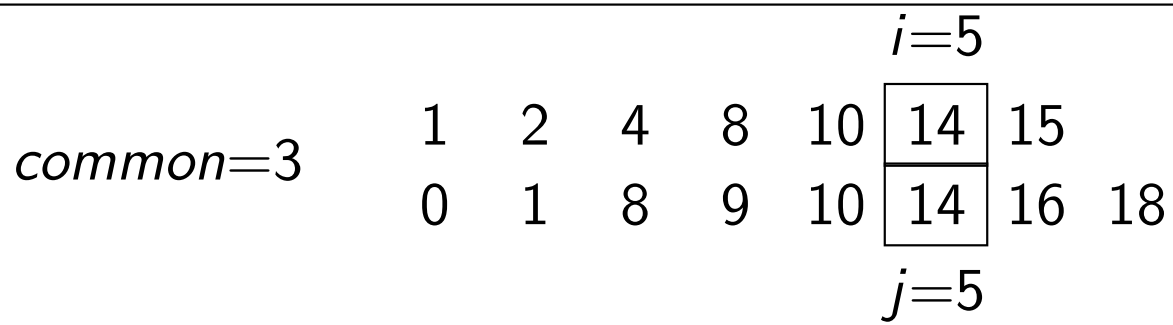


Count common elements in sorted lists of integers

$$ulist[i] < vlist[j] \;\Rightarrow\; i \leftarrow i + 1$$

$$ulist[i] > vlist[j] \;\Rightarrow\; j \leftarrow j + 1$$

$$\boxed{ulist[i] = vlist[j] \;\Rightarrow\; (i, j, common) \leftarrow (i + 1, j + 1, common + 1)}$$

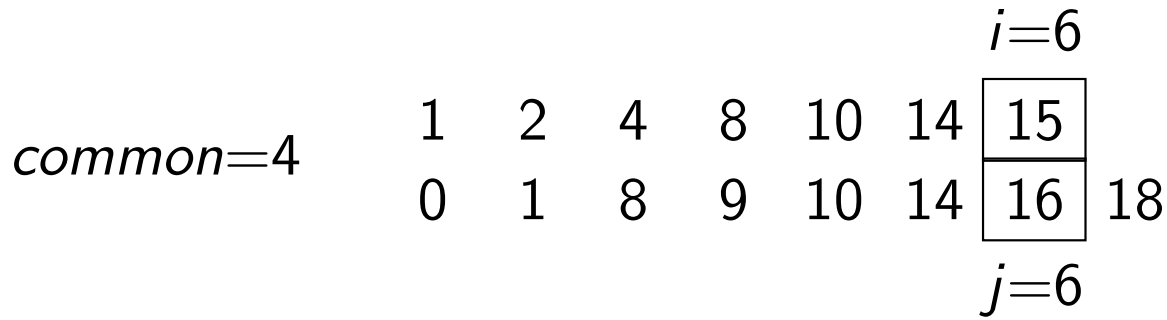


Count common elements in sorted lists of integers

$$ulist[i] < vlist[j] \;\Rightarrow\; i \leftarrow i + 1$$

$$ulist[i] > vlist[j] \;\Rightarrow\; j \leftarrow j + 1$$

$$ulist[i] = vlist[j] \;\Rightarrow\; (i, j, common) \leftarrow (i + 1, j + 1, common + 1)$$

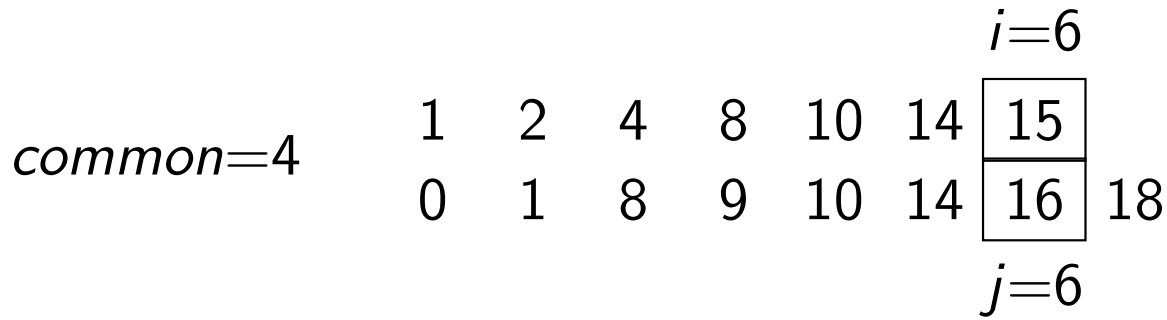


Count common elements in sorted lists of integers

$$\boxed{ulist[i] < vlist[j] \;\Rightarrow\; i \leftarrow i+1}$$

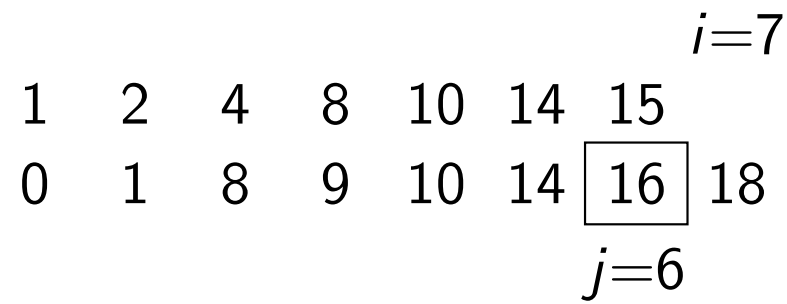
$$ulist[i] > vlist[j] \;\Rightarrow\; j \leftarrow j+1$$

$$ulist[i] = vlist[j] \;\Rightarrow\; (i, j, common) \leftarrow (i+1, j+1, common+1)$$



Count common elements in sorted lists of integers

$$ulist[i] < vlist[j] \;\Rightarrow\; i \leftarrow i + 1$$
$$ulist[i] > vlist[j] \;\Rightarrow\; j \leftarrow j + 1$$
$$ulist[i] = vlist[j] \;\Rightarrow\; (i, j, common) \leftarrow (i + 1, j + 1, common + 1)$$



algorithm can be generalized to handle duplicated elements:
if the same element occurs g times in *ulist* and h times in *vlist*, add $g \cdot h$ to *common*

Mash: the basic idea

- for the comparison of two single sequences u and v, the previous method is considerably faster than the algorithm computing the q-gram distance
- but if there are hundreds or thousands of sequences to be compared (all-against-all), the method is likely not efficient enough (especially in terms of space requirement)
- in such an application scenario, Mash (Ondov et. al.) considerably reduces the runtime and space requirement

Mash (Ondov et. al.)

1. computes reduced representation of q-grams of sequence (sketch)
2. *estimates* Jaccard Index from sketches of the sequences

   (Jaccard estimate is often close to the Jaccard Index)
3. derives distance measure from Jaccard estimate

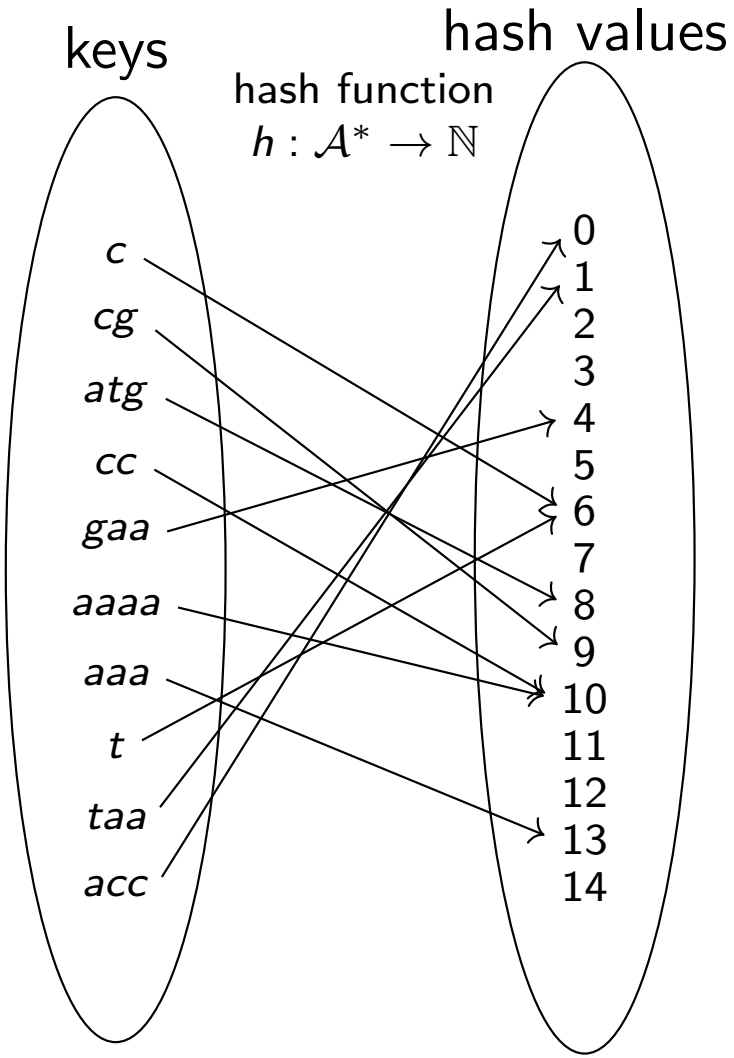
Excursion to hash functions

- one of the very basic tasks in computer science is to efficiently store values associated with keys
- simple solution:
  - store key/value pairs in list and use linear search to find value for given key in $O(n)$ time and space for n key value pairs
  - sort the keys and use binary search to find the value for a given key in $O(\log n)$ time and $O(n)$ space
- one usually wants to access the value for any key in $O(1)$ time
- a very common way to achieve this is to uniquely associate a key with an index of an array where to store the value
- this association is established by a hash function:

  a hash function maps any kind of (hashable) object to a unique integer

- see example for string-keys below

Excursion to hash functions



- collision when $h(w) = h(w')$ for $w \neq w'$, as in $h(cc) = 10 = h(aaaa)$ or $h(c) = 6 = h(t)$
- strategies to solve such conflicts: hashing with chaining, double hashing, open addressing, cuckoo hashing ...
- a hash function is used in a Python-dictionary or a Ruby-Hash or a map in the C++-standard template library
- it is hidden from the user
- Python: obtain hash-value via method `hash`, e.g. `hash('atcg')` $\Rightarrow$ 1 231 534 521 241 347 127
- can be applied to any hashable object (e.g. strings, numbers, functions)

Examples of hash functions for strings

$$\text{js}(s) = h_1(s, |s|) \text{ where}$$

$$h_1(s,i) = \begin{cases} 0 & \text{if } i = 0 \\ (ord(s[i]) + h_1(s, i-1) \cdot 2^5 + h_1(s, i-1)/4) & \text{otherwise} \\ \quad \hat{} \; h_1(s, i-1) & \end{cases}$$

$$\text{sdbm}(s) = h_2(s, |s|) \text{ where}$$

$$h_2(s,i) = \begin{cases} 0 & \text{if } i = 0 \\ ord(s[i]) + h_2(s, i-1) \cdot (2^6 + 2^{16} - 1) & \text{otherwise} \end{cases}$$

$$bp(s) = h_3(s, |s|) \text{ where}$$

$$h_3(s,i) = \begin{cases} 0 & \text{if } i = 0 \\ ord(s[i]) \; \hat{} \; (h_3(s, i-1) \cdot 2^7) & \text{otherwise} \end{cases}$$

- *ord* maps characters to integers
- $\hat{}$ stands for exclusive or

Mash: sketch

Input

- input sequence u
- length q
- sketch size $s \geq 1$
- hash-function $h : \mathcal{A}^q \to \mathbb{N}$

$$u = \texttt{gtgatgtagtgagaagtag}$$
$$\Downarrow \quad q = 5$$
$$G_q(u) = \left\{ \begin{array}{l} \texttt{gtgat gtagt gagaa} \\ \texttt{tgatg tagtg agaag} \\ \texttt{gatgt agtga gaagt} \\ \texttt{atgta gtgag aagta} \\ \texttt{tgtag tgaga agtag} \end{array} \right\}$$

Algorithm

- enumerate set $G_q(u)$ of all q-grams of u
- compute $H_{q,h}(u) = \{h(w) \mid w \in G_q(u)\}$
- keep only the s smallest values of $H_{q,h}(u)$
- this is the sketch of u, denoted by $S_{q,h,s}(u)$
- fix q, h, s and abbreviate $S_{q,h,s}(u)$ by $S(u)$

$$\Downarrow \quad h$$
$$H_{q,h}(u) = \left\{ \begin{array}{r} 26\ 35\ 62 \\ 78\ 56\ 33 \\ 34\ \ 6\ 44 \\ 9\ 64\ 94 \\ 66\ 10\ 72 \end{array} \right\}$$
$$\Downarrow \quad s = 5$$
$$S(u) = \{6, 9, 10, 26, 33\}$$

Mash: hash functions

- requirement for hash-function: avoid collisions, i.e. different q-grams w and t satisfying

$$h(w) = h(t)$$

- q-gram encoding, as used for q-gram distance has no collisions
- but when encoding $\mathtt{A} \mapsto 0$, $\mathtt{C} \mapsto 1$, $\mathtt{G} \mapsto 2$, $\mathtt{T} \mapsto 3$, poly-$\mathtt{A}$ sequences get smallest hash-values and more likely are represented in sketches
- there are other encodings, which avoid such effect, but require slightly more runtime to compute

MinHash

- the fact that the smallest s hash-values form the sketch leads to the name of the technique: MinHash[a]
- for appropriate hash-functions $S(u)$ is a random sample of u
- random samples allow to derive an estimate of the Jaccard Index

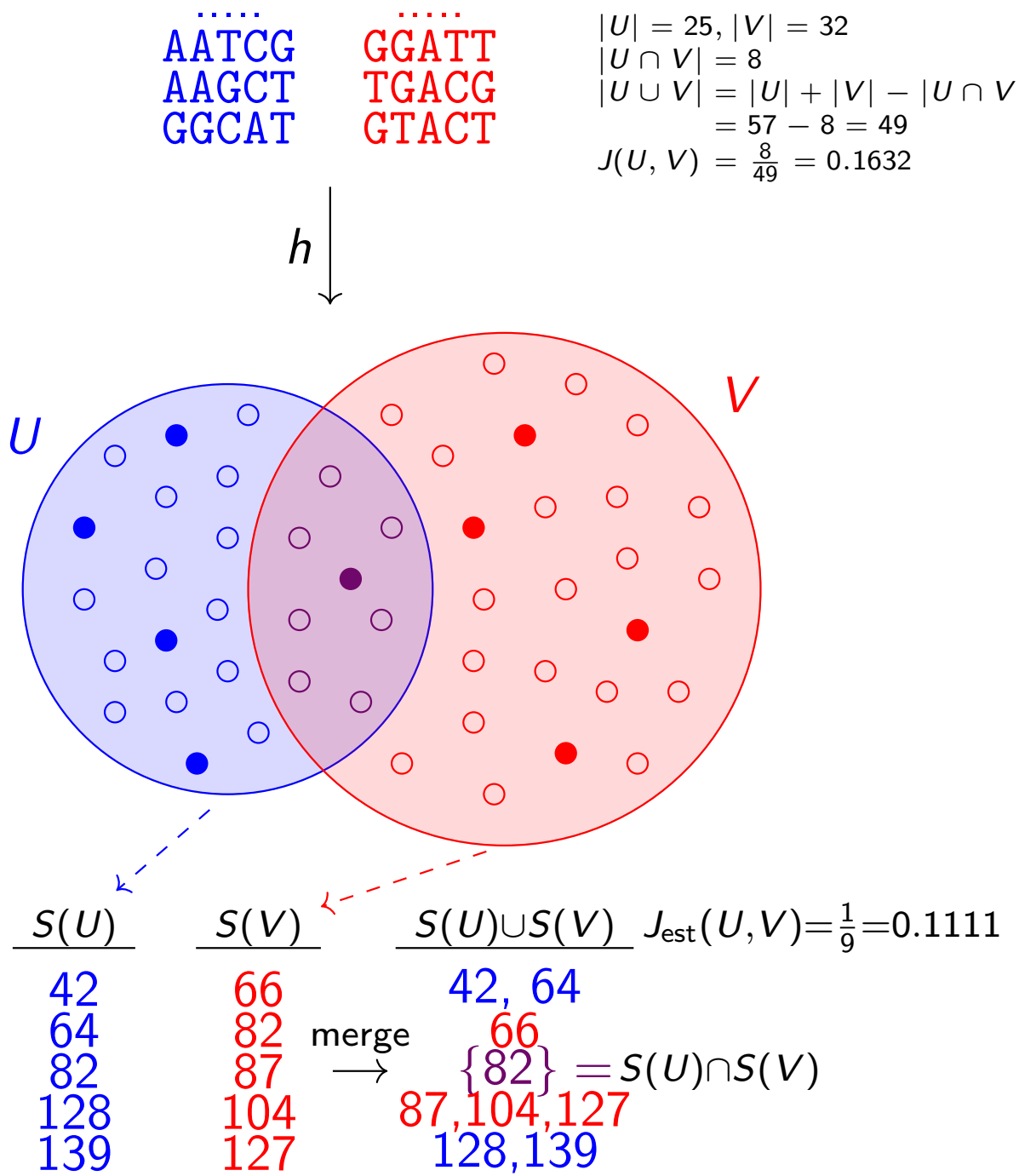
[a] Broder, 1997, where this technique was used for identifying duplicate documents on the WEB

Idea

as $S(U)$ and $S(V)$ are random samples of the q-grams of u and v, the fraction of the shared items in $S(U)$ and $S(V)$ is expected to be similar as in U and V

Jaccard estimate

$$J_{\text{est}}(U,V) = \frac{|S(U) \cap S(V)|}{|S(U) \cup S(V)|}$$



Computation of Jaccard estimate $J_{\text{est}}(U, V)$

Computation of $S(U)$ for $U = G_q(u)$ with $m = |u|$

- enumerate all q-grams of u: $O(m)$ runtime, $O(q)$ space
- apply hash function f to each q-gram: $O(mq)$ runtime, $O(1)$ space
- use min-heap to keep s smallest hash-values seen so far: $O(m \log s)$ runtime, $O(s)$ space

$\Rightarrow$ $O(mq + m \log s) = O(m(q + \log s))$ runtime, $O(s)$ space

Computation of $S(V)$ for $V = G_q(v)$ with $n = |v|$

$\Rightarrow$ $O(n(q + \log s))$ runtime, $O(s)$ space

Computation of Jaccard estimate $J_{\text{est}}(U, V)$

Computation of $|S(U) \cap S(V)|$

- merge 2 sorted lists of integers of length s to count common elements: $O(s)$ runtime, $O(1)$ space

Computation of $|S(U) \cup S(V)|$

- compute $|S(U)| + |S(V)| - |S(U) \cap S(V)|$ in constant time

Total runtime

- $O(\underbrace{m(q + \log s)}_{\text{sketch of } u} + \underbrace{n(q + \log s)}_{\text{sketch of } v} + \underbrace{s}_{\textit{merge}}) = O((m + n)(q + \log s) + s)$ runtime

Total Space requirement

- $O(s)$

- compare to computation of q-gram distance in $O(m + n + r^q)$ runtime and space
- MinHash-based method does not have improved running time, but uses less space

All-against-all comparison via Jaccard estimate

All-against-all

- given set $T = \{t_1, t_2, \ldots, t_k\}$ of k sequences
- all-against-all comparison: compare all pairs t_i and t_j, $1 \leq i < j \leq k$

Jaccard estimate for all pairs t_i, t_j

1. precompute sketches of all t_i and store each on file in sorted order
   $O(|t_1|(q+\log s)+|t_2|(q+\log s)+\cdots+|t_k|(q+\log s)) = O(n(q+\log s))$ time, where $n = \sum_{i=1}^{k} |t_i|$
2. compute Jaccard estimate for each pair t_i, t_j, $i < j$ in $O(s)$ runtime: $O(k^2 s)$ runtime for all pairs

$O(n(q + \log s) + k^2 s)$ total runtime $\Rightarrow$ very fast, as only step 1 depends on length of sequences

In the following we will show that the Jaccard estimate allows to derive an approximation of the mutation rate of the sequences to be compared

Excursion to Poisson Distributions

Definition 1 (Poisson Distribution)

A Poisson Distribution P_λ is a discrete probability distribution with a parameter λ (describing the average number of events per interval) defined by

$$P_\lambda(k \text{ events in interval}) = \mathrm{e}^{-\lambda}\frac{\lambda^k}{k!}$$

with $k \in \mathbb{N}$ and $\mathrm{e} \approx 2.71828$ is the Eulerian Number.

A Poisson distributions is used to quantify the probability of events which occur in a period of time or a geographical location, if the following holds:

- k is the number of times an event occurs in an interval
- the events occur independently
- the rate at which the events occur is constant
- two events never occur at the same sub-interval or sub-location

Excursion to Poisson Distributions

Applications of Poisson Distributions

- Telecommunication: telephone calls arriving in a system
- Astronomy: photons arriving at a telescope
- Management: customers arriving at a counter
- Finance: number of losses occurring in a given period of time
- Land management: number of overfloods at a coast line
- Earthquake seismology: seismic risk for large earthquakes
- Radioactivity: number of decays in a given time interval in a radioactive sample
- Text editing: number of typos on a printed page
- Biology: number of mutations on a strand of DNA per unit length

from `https://en.wikipedia.org/wiki/Poisson_distribution`

Poisson distributions for modelling DNA-mutations

- consider a DNA sequence in which we have on average 1 mutation per q-gram for $q = 18$
- so our interval size is 18 and $\lambda = 1$
- assuming a Poisson distribution we calculate

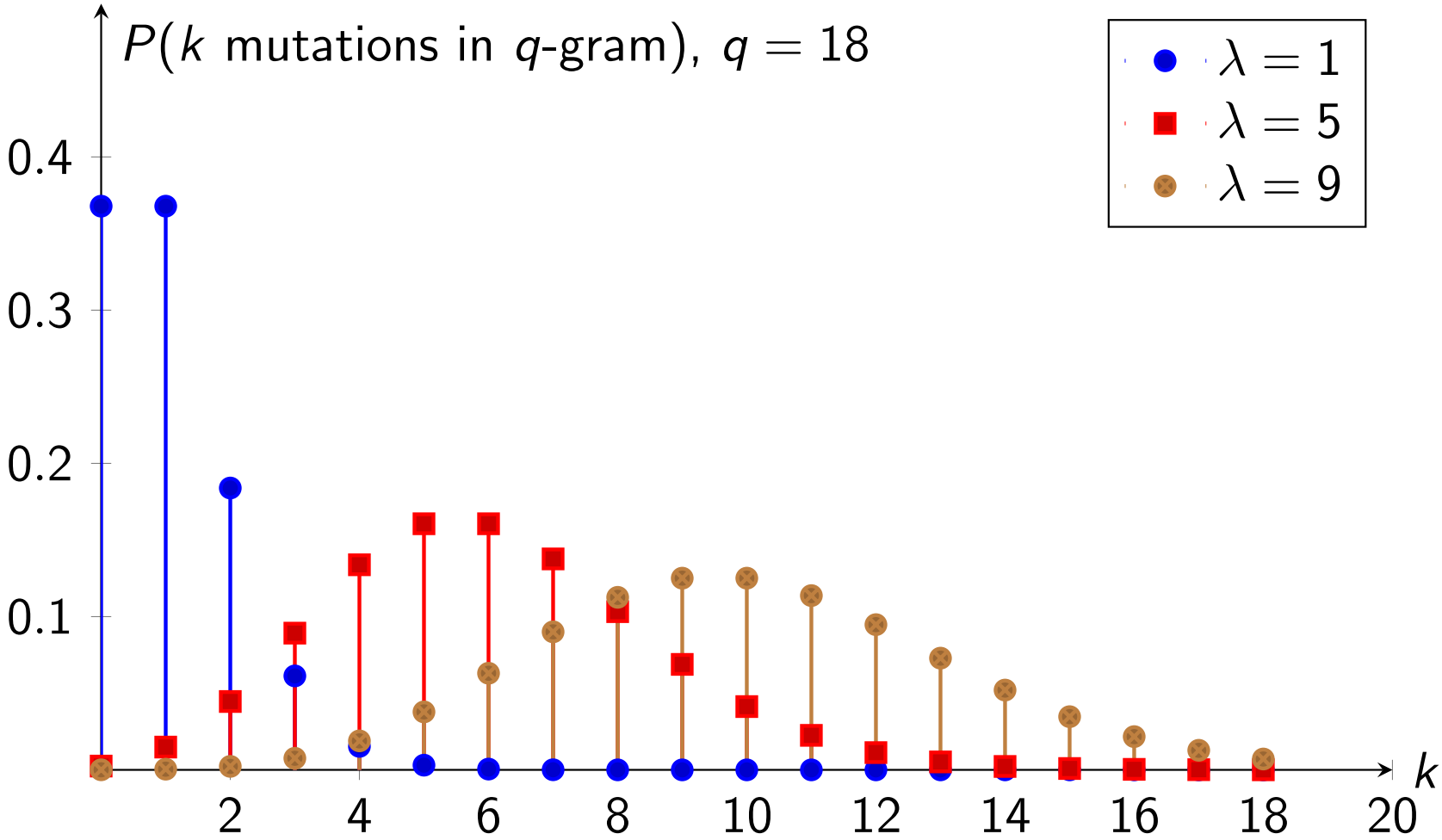
$$P_\lambda(k \text{ mutations in } q\text{-gram}) = \mathrm{e}^{-\lambda}\frac{\lambda^k}{k!} = \mathrm{e}^{-1}\frac{1^k}{1!}$$

$$P_\lambda(0 \text{ mutations in } q\text{-gram}) = \mathrm{e}^{-1}\frac{1^0}{0!} = \mathrm{e}^{-1} \approx 0.368$$

$$P_\lambda(1 \text{ mutations in } q\text{-gram}) = \mathrm{e}^{-1}\frac{1^1}{1!} = \mathrm{e}^{-1} \approx 0.368$$

$$P_\lambda(2 \text{ mutations in } q\text{-gram}) = \mathrm{e}^{-1}\frac{1^2}{2!} = \frac{\mathrm{e}^{-1}}{2} \approx 0.184$$

Poisson distribution for $\lambda \in \{1, 5, 9\}$



from `https://tex.stackexchange.com/questions/282806/plot-the-poisson-function-correctly`

Probability of matching q-grams

Assumption of evolutionary model of sequences

- sequences differ by mutations (indels and mismatches)
- mutations occur at single nucleotides randomly and independently with a constant rate d per base over the entire sequence
- so qd is average mutation rate per q-gram

$\Rightarrow$ probability of events (i.e. mutations) occurring in q-gram of the sequence follows a Poisson distribution with $\lambda = qd$, i.e.

$$P_\lambda(\overbrace{k \text{ mutations in } q\text{-gram}}^{\text{event}}) = \mathrm{e}^{-qd}\frac{(qd)^k}{k!}$$

$$\Rightarrow P_\lambda(\text{matching } q\text{-gram}) = P_\lambda(0 \text{ mutations in } q\text{-gram})$$

$$= \mathrm{e}^{-qd}\frac{(qd)^0}{0!}$$

$$= \mathrm{e}^{-qd}\frac{1}{1} = \mathrm{e}^{-qd}$$

Probability of q-matches and Jaccard Index

- let $n = \frac{|u|+|v|}{2}$ be the average length of u and v
- if e^{-qd} is the probability of a single q-gram match, then we expect

$$\gamma = n \cdot \mathrm{e}^{-qd}$$

  q-gram matches between the two sequences u and v
- Recall: with $U = G_q(u)$ and $V = G_q(v)$ we have

$$J(u, v) = J(U, V) = \frac{|U \cap V|}{|U \cup V|} \approx \frac{\gamma}{2n - \gamma} \tag{1}$$

- In (1) one ignores that q-grams may occur more than once $\Rightarrow$ $\gamma \geq |U \cap V|$ and $2n - \gamma \geq |U \cup V|$
- but the ratio is very similar (so we use the symbol $\approx$ to express this)

Probability of q-matches and Jaccard Index

– In $J(u,v) \approx \frac{\gamma}{2n-\gamma}$ we substitute γ by $n \cdot \mathrm{e}^{-qd}$:

$$
\begin{aligned}
& J(u,v) \approx \frac{n \cdot \mathrm{e}^{-qd}}{2n - n \cdot \mathrm{e}^{-qd}} = \frac{n \cdot \mathrm{e}^{-qd}}{n \cdot (2 - \mathrm{e}^{-qd})} = \frac{\mathrm{e}^{-qd}}{2 - \mathrm{e}^{-qd}} \\
\iff \quad & J(u,v) \cdot (2 - \mathrm{e}^{-qd}) \approx \mathrm{e}^{-qd} \\
\iff \quad & 2 \cdot J(u,v) - J(u,v) \cdot \mathrm{e}^{-qd} \approx \mathrm{e}^{-qd} \\
\iff \quad & 2 \cdot J(u,v) \approx J(u,v) \cdot \mathrm{e}^{-qd} + \mathrm{e}^{-qd} \\
\iff \quad & 2 \cdot J(u,v) \approx (J(u,v) + 1) \cdot \mathrm{e}^{-qd} \\
\iff \quad & \frac{2 \cdot J(u,v)}{J(u,v) + 1} \approx \mathrm{e}^{-qd}
\end{aligned}
$$

Probability of q-matches and Jaccard Index

– switching the left and right-hand size of this (approximate) equation, we obtain

$$
\begin{aligned}
& \mathrm{e}^{-qd} \approx \frac{2 \cdot J(u,v)}{J(u,v)+1} \\
\iff \quad & \ln \mathrm{e}^{-qd} \approx \ln \frac{2 \cdot J(u,v)}{J(u,v)+1} \\
\iff \quad & -qd \approx \ln \frac{2 \cdot J(u,v)}{J(u,v)+1} \\
\iff \quad & d \approx -\frac{1}{q} \ln \frac{2 \cdot J(u,v)}{J(u,v)+1}
\end{aligned}
$$

– so we can estimate the mutation rate d (which is what we are interested in) in terms of $J(u,v)$ and q

Mutation rate and Jaccard Estimate

Now the Jaccard estimate comes back into play:

$$\text{as } J_{\text{est}}(u,v) = J_{\text{est}}(U,V) = \frac{|S(U) \cap S(V)|}{|S(U) \cup S(V)|} \approx J(u,v)$$

$$\text{we conclude } d \approx -\frac{1}{q} \ln \frac{2 \cdot J_{\text{est}}(u,v)}{J_{\text{est}}(u,v)+1}$$

– the mash distance $MD_{q,s}(u,v)$ derives the mutation rate of u and v from the Jaccard estimate of u and v:

$$MD_{q,s}(u,v) = \begin{cases} 1 & \text{if } J_{\text{est}}(u,v) = 0 \\ -\frac{1}{q} \ln \frac{2 \cdot J_{\text{est}}(u,v)}{J_{\text{est}}(u,v)+1} & \text{otherwise} \end{cases}$$

Just like $J_{\text{est}}(u,v)$, $MD_{q,s}(u,v)$ can be computed in $O(s)$ space and $O((|u|+|v|)(q+\log s))$ time

Gold standard for genome distance

- to determine whether the mash distance is a reliable distance estimator, one needs to compare it to a gold standard, i.e. a widely trusted measure of the distance of two (genome) sequences
- such a gold standard is based on the *average nucleotide identity* of u and v (abbreviation: $ANI(u, v)$)
- $ANI(u, v)$ is determined from a set $SLA(u, v)$ of significant local alignments of u and v:

$$ANI(u, v) = \frac{1}{|SLA(u, v)|} \sum_{A \in SLA(u,v)} \left(1 - \frac{2 \cdot \delta(A)}{|A.u| + |A.v|}\right) \tag{2}$$

where

- δ is the unit cost function,
- $A.u/A.v$ are the substrings of u/v involved in alignment A
- so ratio in (2) is the relative number of errors in alignment

Gold standard for genome distance

Computing $SLA(u, v)$

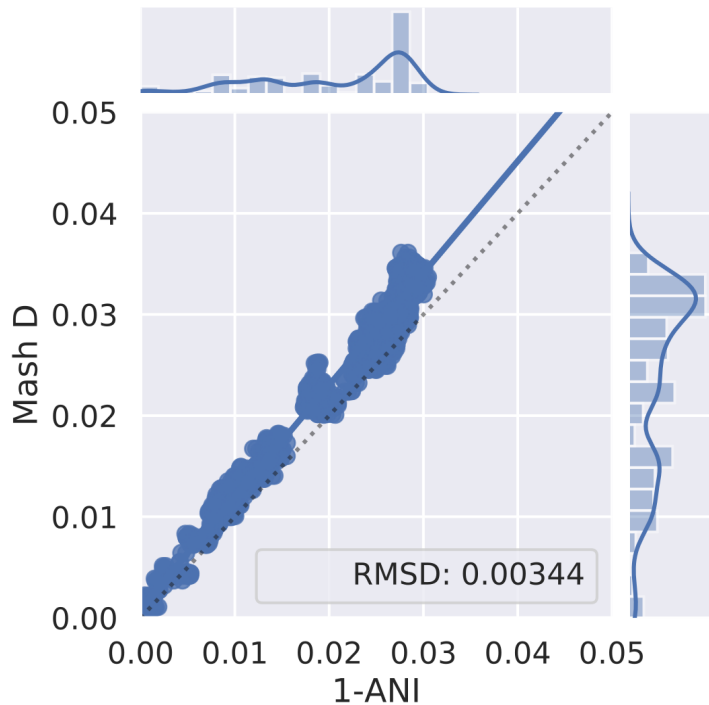
- $SLA(u, v)$ can be computed by any suitable program delivering local alignments
- as the DP-based Smith-Waterman algorithm is often too slow, one resorts to seed-extend methods (see section on Fasta and Blast) to compute local alignments
- the program `nucmer` and `delta-filter` from the MUMmer-software (Kurtz et al. 2004) are widely used in this context (and was applied in the results presented below)
- computations using `nucmer`/`delta-filter` still take much longer than methods not based on alignments

Results 1: verification against gold standard (1/2)

- input data: 500 *Escherichia* genomes $\Rightarrow 500 \cdot (500-1)/2 = 124\,750$ sequence pairs
- for each pair of sequences u and v plot a dot at coordinate $(1 - ANI(u,v), MD_{q,s}(u,v))$, i.e. $1 - ANI(u,v)$ is plotted on X-axis and $MD_{q,s}(u,v)$ is plotted on Y-axis $\Rightarrow$ scatter plot

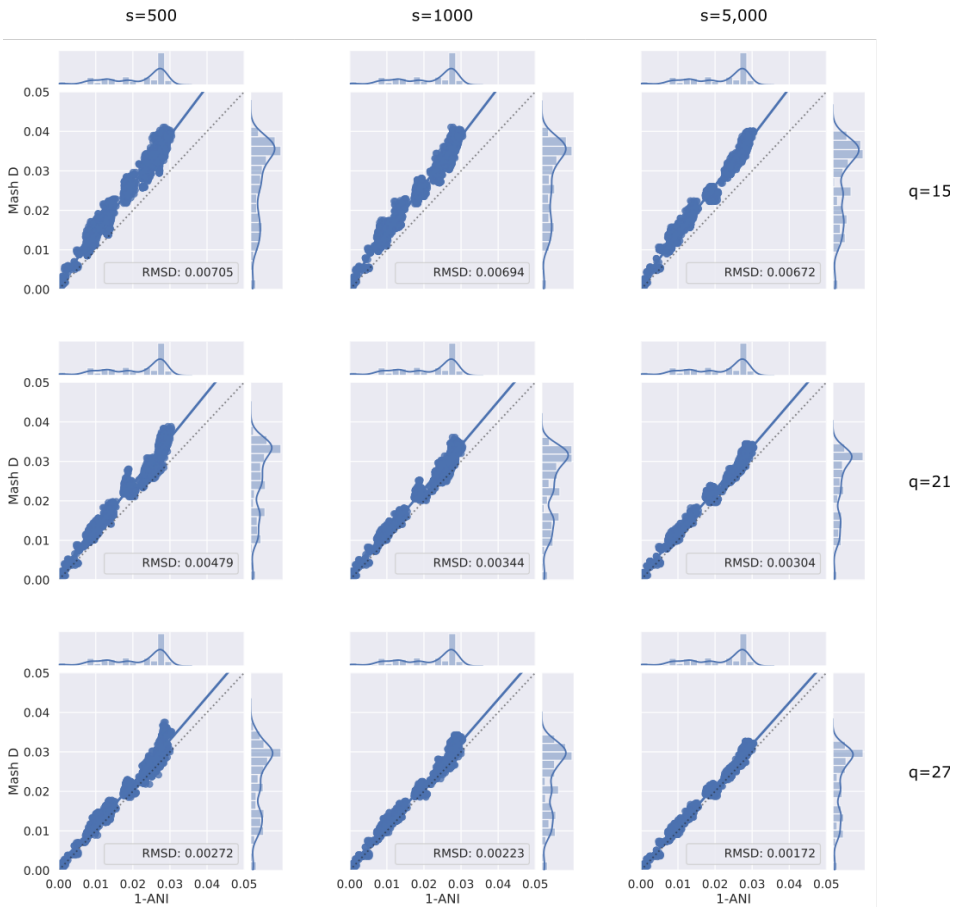
- ideally $\frac{MD_{q,s}()}{1-ANI()} = 1 \Rightarrow$ all values on dotted line
- blue lines: result of a linear regression of the dot-coordinates
- RMSD measures deviation of vectors of $MD_{q,s}()$- and $(1 - ANI())$-values
- histogram on top: distribution of $(1 - ANI())$-values
- rotated histogram on the right: distribution of $MD_{q,s}()$-values

result for $q = 21$, $s = 1\,000$



Results 1: verification against gold standard (2/2)

- rows: scatter plots for constant values of $q = 15$ (first row), $q = 21$ (second row) and $q = 27$ (third row)
- columns: scatter plots for constant values of $s = 500$ (1. col.), $s = 1\,000$ (2. col.) and $s = 5\,000$ (3. col.)



	RMSD from plots		
	s		
q	500	1 000	5 000
15	.00705	.00694	.00672
21	.00479	.00344	.00603
27	.00272	.00223	.00172

- $s \uparrow \Rightarrow$ RMSD $\downarrow$ for $q = 15, 27$
- $q \uparrow \Rightarrow$ RMSD $\downarrow$

$\Rightarrow$ $MD_{q,s}()$ is a very good estimator for distances of closely related genomes

Results 2: Clustering all RefSeq sequences (Ondov et al.)

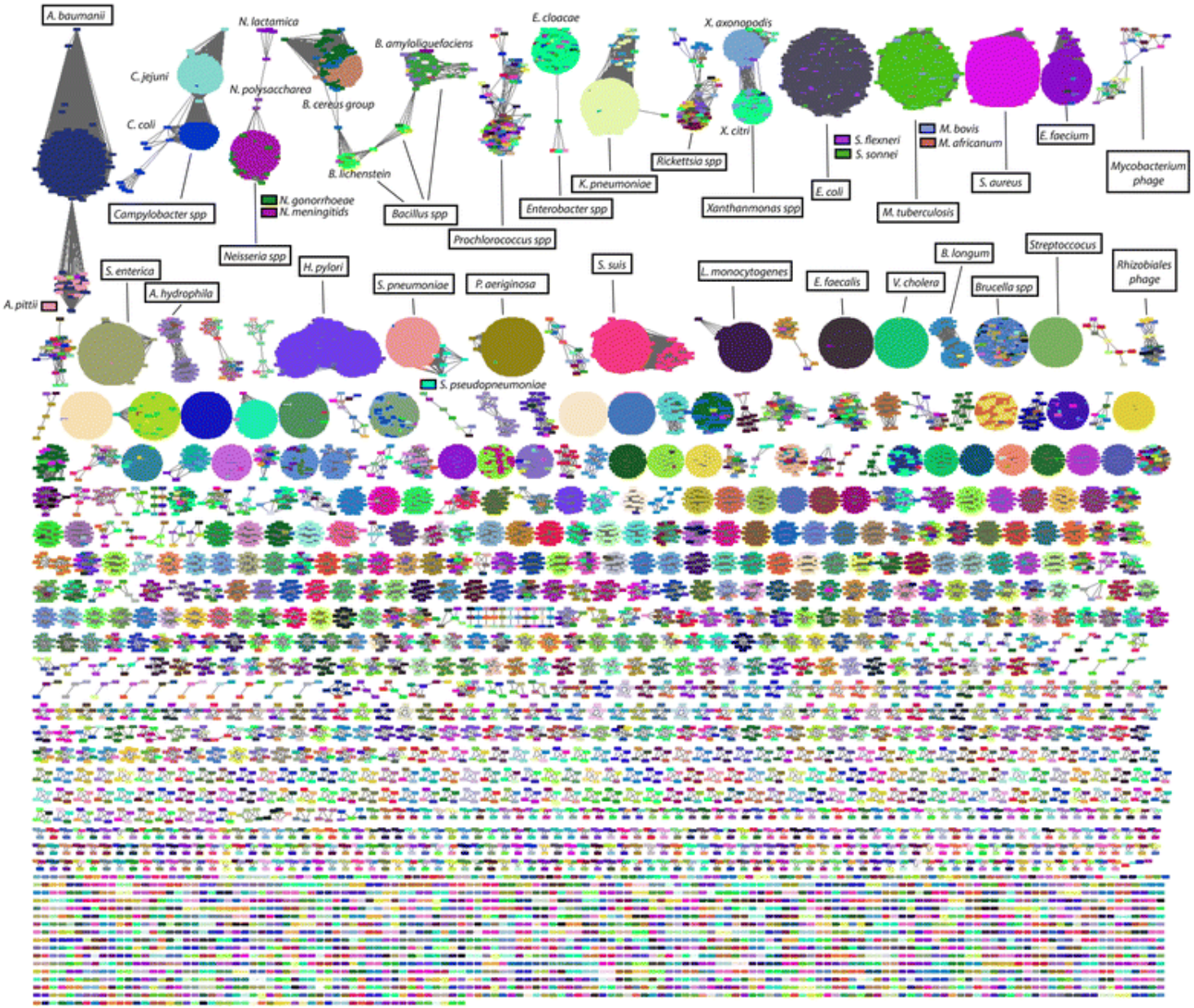
- input: all genomes in NCBI RefSeq Release 70, i.e. 618 GB of genomic sequences from 54 118 organisms
- compute sketches for $q = 16$ and $s = 1\,000$

$\Rightarrow$ total size of all sketches: 93 MB (i.e. 0.091 GB)

- space reduction by factor $\geq 6\,800$
- runtime for computing sketches: 26.1 CPU hours
- $\frac{54\,118 \cdot (54\,118-1)}{2} \approx 1.46 \cdot 10^9$ pairwise comparisons in 6.9 CPU hours
- link all genomes u and v if $MD_{q,s}(u, v) \leq 0.05$ and p-value[3] of the distance is $\leq 1.0e-10$
- resulting clustering, restricted to bacteria and viruses, is shown on Figure 1
- genomes belonging to same taxonomic group form (in most cases) own clusters $\Rightarrow$ consistency with taxonomy

[3] method for computing this is described by Ondov et. al.

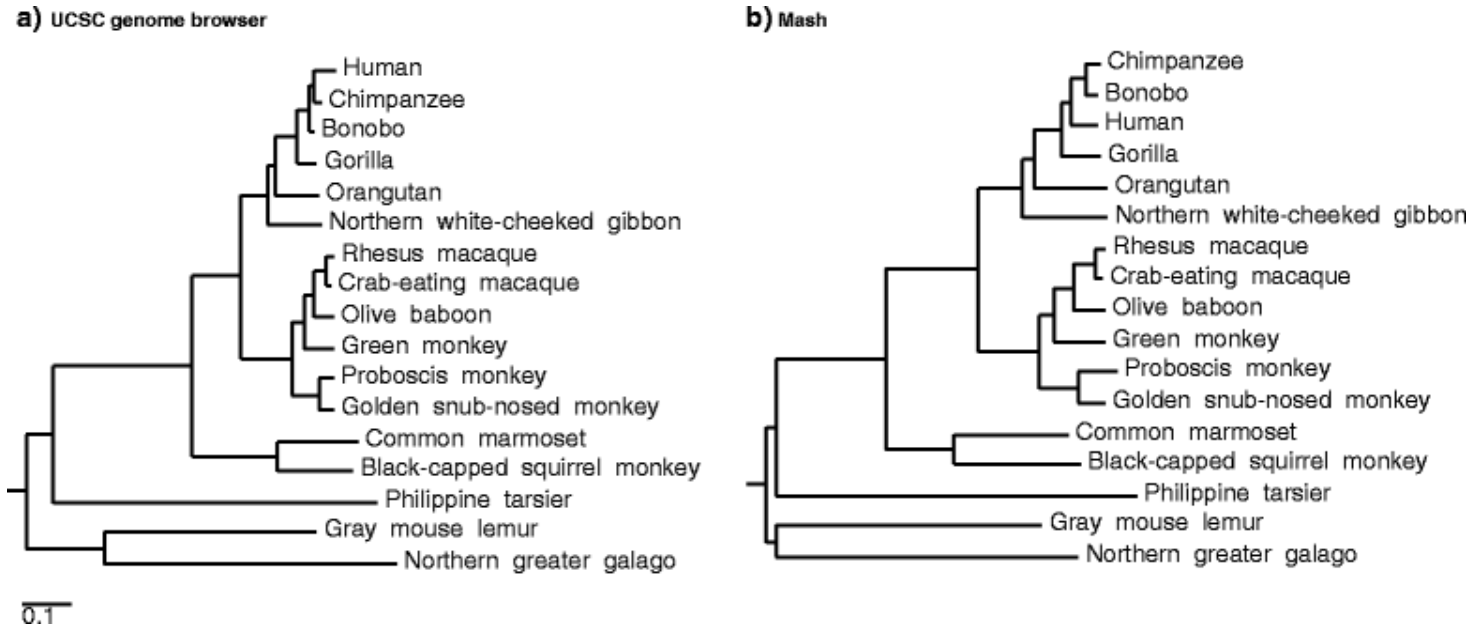
Figure 1: Result of clustering bacteria and viruses using `mash`



Results 3: Clustering 17 primate genomes (Ondov et al.)

- input: 17 RefSeq primate genomes, each of length in the range from $2.6 \cdot 10^9$ to $3.2 \cdot 10^9$ bp
- compute mash distances in 2.5 CPU h for $s = 1\,000$ and $q = 21$
- from mash distances construct a phylogenetic tree using the Neighbor-Joining algorithm (discussed in the section on Phylogeny)
- compare it with alignment-based phylogenetic tree model downloaded from the UCSC genome browser, see Figure 2.

Figure 2: Phylogeny of 17 primate genomes: a) from UCSC genome browser and b) determined based on distances computed by `mash`



- topologically, both phylogenies are consistent, except for the split of Human versus Chimpanzee/Bonobo
- in this branch the mash topology is more similar to past phylogenetic studies and mitochondrial trees
- on average, branch lengths of the mash-based tree are slightly longer

Impact of the MinHash concept in genome informatics

- since 2015 the MinHash concept was used in several improved methods for solving important problems in large scale sequence comparison:
  - identification of overlaps in long reads for sequence assembly by MHAP [Berlin et al., 2015],
  - mapping of long reads to reference sequences by MashMap [Jain et al., 2017],
  - computing homology-maps for complete genomes [Jain et al., 2017],
  - estimation of containments of long reads by Mash-Screen [Ondov et al., 2019],
  - all-against-all comparison of 90 000 prokaryotic genomes [Jain et al., 2018],
  - weighted minimizer sampling for mapping long reads in repetitive genomes [Jain et al., 2020],
  - k-mer counting [Deorowicz et al., 2015, Kokot et al., 2017].
- the review [Marçais et al., 2019, page 108-113] gives an overview of the theory and of applications of the MinHash concept

Conclusion

- sketches are constant size representations of samples of q-grams from a set of sequences
- for a sequence of length n, its sketch of size s can be computed in $O(n(q + \log s))$ time
- sketches are random samples and can be used to estimate the Jaccard Index of the sets of all q-grams of two sequences
- assuming the mutations are Poisson-distributed, one can show that the Jaccard Index is related to the mutation rate
- the mutation rate can thus be approximated by the Jaccard estimate to obtain the Mash distance $MD_{q,s}(u, v)$
- the evolutionary model based on Poisson processes is very simple and does not attempt to model more complex evolutionary processes
- there are some interesting applications of the Mash distance, such as large scale clustering of sequence sets or construction of phylogenies
- Ondov et. al. report more applications, like real-time genome identification from assemblies or reads and clustering massive metagenomic datasets


Berlin, K., Koren, S., Chin, C.-S., Drake, J., Landolin, J. M., and Phillippy, A. M. (2015).
Assembling large genomes with single-molecule sequencing and locality sensitive hashing.
Nat. Biotech., 33(6).


Deorowicz, S., Kokot, M., Grabowski, S., and Debudaj-Grabysz, A. (2015).
KMC 2: fast and resource-frugal k-mer counting.
Bioinformatics, 31(10):1569–1576.


Jain, C., Dilthey, A., Koren, S., Aluru, S., and Phillippy, A. M. (2017).
A fast approximate algorithm for mapping long reads to large reference databases.
In *International Conference on Research in Computational Molecular Biology*, pages 66–81. Springer.


Jain, C., Rhie, A., Zhang, H., Chu, C., Walenz, B. P., Koren, S., and Phillippy, A. M. (2020).
Weighted minimizer sampling improves long read mapping.

Bioinformatics (Oxford, England), 36(1):i111–i118.


Jain, C., Rodriguez-R, L. M., Phillippy, A. M., Konstantinidis, K. T., and Aluru, S. (2018).
High throughput ANI analysis of 90K prokaryotic genomes reveals clear species boundaries.
Nature communications, 9(1):5114.


Kokot, M., Długosz, M., and Deorowicz, S. (2017).
Kmc 3: counting and manipulating k-mer statistics.
Bioinformatics, 33(17):2759–2761.


Marçais, G., Solomon, B., Patro, R., and Kingsford, C. (2019).
Sketching and Sublinear Data Structures in Genomics.
Annual Review of Biomedical Data Science, 2(1):93–118.


Ondov, B. D., Starrett, G. J., Sappington, A., Kostic, A., Koren, S., Buck, C. B., and Phillippy, A. M. (2019).
Mash screen: high-throughput sequence containment estimation for genome discovery.
Genome Biology, 20(1):232.


Ondov, B. D., Treangen, T. J., Melsted, P., Mallonee, A. B., Bergman, N. H., Koren, S., and Phillippy, A. M. (2016).
Mash: fast genome and metagenome distance estimation using minhash.
Genome Biology, 17(1):1–14.