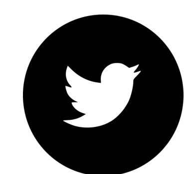
On Weighted *k*-mer Dictionaries

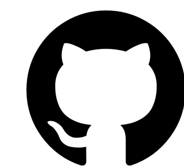
Paper: https://doi.org/10.4230/LIPIcs.WABI.2022.9
Code: https://github.com/jermp/sshash

Giulio Ermanno Pibiri

Ca' Foscari University of Venice and ISTI-CNR



@giulio_pibiri



@jermp

WABI 2022 (The 22-nd International Workshop on Algorithms in Bioinformatics)
Potsdam, Germany, 5-9 September 2022

The Weighted *k*-mer Dictionary Problem

- We are given a large string S over the alphabet {A,C,G,T} (e.g., a genome or a pan-genome). Let $K = \{\langle g, w(g) \rangle \,|\, g \in S\}$, where g is a k-mer and $w(g)$ is the number of occurrences — the *weight* — of g.

  Example: The human genome (GRCh38) has >2.5B distinct k-mers for $k = 31$.

- **Problem.** We want to build a dictionary for K so that the following operations are efficient:
  - Lookup$(g) = i$, where $1 \leq i \leq n$ if $g \in S$ or $i = -1$ otherwise;
  - Access$(i) = g$ if $1 \leq i \leq n$;
  - Count$(g) = w(g)$ if $g \in S$.

  (Other operations of interest are *iteration, streaming* queries, and *navigational* queries.)

A World of *k*-mer Indexes

- Huge research effort produced many types of indexes based on k-mers, with different:

  - representations (hashing, BWT-based, exact vs. approximate),
  - features (e.g., static vs. dynamic),
  - space/time trade-offs,
  - operations, etc.

- Recent surveys on this topic:

  - Data Structures based on *k*-mers for Querying Large Collections of Sequencing Data Sets
    Marchet et al., Genome Research, 2020.

  - Data Structures to Represent a Set of *k*-long DNA Sequences
    Chikhi et al., ACM Computing Surveys, 2021.

Sparse and Skew Hashing (SSHash)

Bioinformatics/ISMB 2022

- Our focus is on hash-based data structures, for their fast query evaluation.

- In a prior work, we presented *SSHash* — a fast and compact k-mer dictionary based on minimal perfect hashing.

- Algorithmic tool-box: spectrum-preserving string sets, minimizers, minimal perfect hashing, compressed encodings (e.g., Elias-Fano).

- Code in C++17 is available at: https://github.com/jermp/sshash.

- However, we did not consider the weights of the k-mers.

Weighted SSHash (w-SSHash)

- This work: enrich SSHash with the weights.
- Questions:

  - **Q1.** What is the surplus in index space for the weights?

  - **Q2.** How well can the weights be compressed?

  - **Q3.** Do they have an impact on query time (Lookup vs. Count)?

SSHash is Order-Preserving

- **Recap.** For the n distinct k-mers of S, SSHash implements a function (Lookup) $h : \Sigma^k \rightarrow \{-1, 1, \ldots, n\}$, where $1 \le h(g) \le n$ if $g \in S$ and $h(g) = -1$ if $g \notin S$.
- So the hash code $h(g) = i$ can be directly used to associate some satellite information to the k-mer g.
  For example, its weight: $W[1..n]$ where $W[i = h(g)] = w(g)$.
- **Order-Preserving Property.** If g_2 if the successor of g_1, then: $h(g_2) = h(g_1) + 1$.
- This is a direct consequence of indexing a spectrum-preserving string set (SPSS): S is reduced to a set of $m \ge 1$ strings $\mathcal{S} = \{S_1, \ldots, S_m\}$.
- Any order on $\mathcal{S}$ uniquely determines an order $i = 1, \ldots, n$ for the k-mers g_i, thus: $h(g_i) = i$.

1:

5	5	2	...	2	2		
A	C	C	...	G	T	G	T

2:

1	1	2	...	2	2		
C	T	T	...	C	A	T	T

3:

3	3	3	...	2	2		
C	G	A	...	T	T	T	C

4:

3	3	1	...	1	1		
G	A	T	...	C	C	G	A

Example $\mathcal{S}$ with $m = 4$.

The Weights

- Let $W[1..n]$ be the sequence of k-mer weights, where $W[i] = w(g_i)$ and $i = h(g_i)$.

- **Empirical Property.** Consecutive k-mers in $\mathcal{S}$ (the SPSS) are likely to have the **same** weight.

- **Order-Preserving Property.** If g_2 if the successor of g_1, then: $h(g_2) = h(g_1) + 1$.

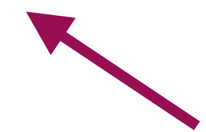
- These two properties → W has **runs** of equal weights.
  We exploit the order of the k-mers to *preserve* the natural order of the weights.

```
>5 5 5 5 5 5 5 5 5 5 5 5 5 5
GGTAATGCAGCCAGGGATGCAACGACCGCAACAGAAAAAGCCCG

>4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 4 4 4 4
CAGCTCATTACAGAAAAAATACCGCTCACCGCCCTGCACCGTCAGGTCAATTTCCCTGAGCACCACCCGCGGTGACTGCTCTGATTTAACC

>4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
CAGCTATGCAGGAGACAAGAATCGCCAGCTTACCCGTTACAGCGATACCCGCTGGCATG

>13 13 13 13 13 13 13
TCAGGTGTACGGTGTGCGTAAAGTCTGGCGTCAGTTG
```



We have 6 runs in this example (k=31):
5 (14x), **4** (18x), **2** (8x), **1** (31x), **4** (33x), **13** (7x).

Run-Length Encoding (RLE)

- Represent W with r **runs** as a sequence of run-length pairs $RLW = \langle w_1, \ell_1 \rangle \langle w_2, \ell_2 \rangle \ldots \langle w_r, \ell_r \rangle$.
- Take the prefix-sums of the sequence $0, \ell_1, \ell_2, \ldots, \ell_{r-1}$ into an array $L[1..r]$ and encode it with Elias-Fano.
- We spend, at most

$$r \cdot \left(c + \lceil \log_2(n/r) \rceil + 2 + o(1) \right)$$ bits for RLW (on top of SSHash).

Number of runs. Number of bits dedicated to each w_i. Elias-Fano on the lengths.

- To retrieve $w(g)$ from $i = h(g)$: identify the run containing i. All that we need is a *predecessor query* over L which can be done in $O(\log(n/r))$ with Elias-Fano.

Reducing the Number of Runs

- We spend space proportional to the **number of runs** in the weights W.
- So we study the problem of reducing the number of runs to optimise the space.

  Q. What are our degrees of freedom to better compress W ?

  - The **order** of the strings $S_1, \ldots, S_m$ in the SPSS $\mathscr{S}$, and

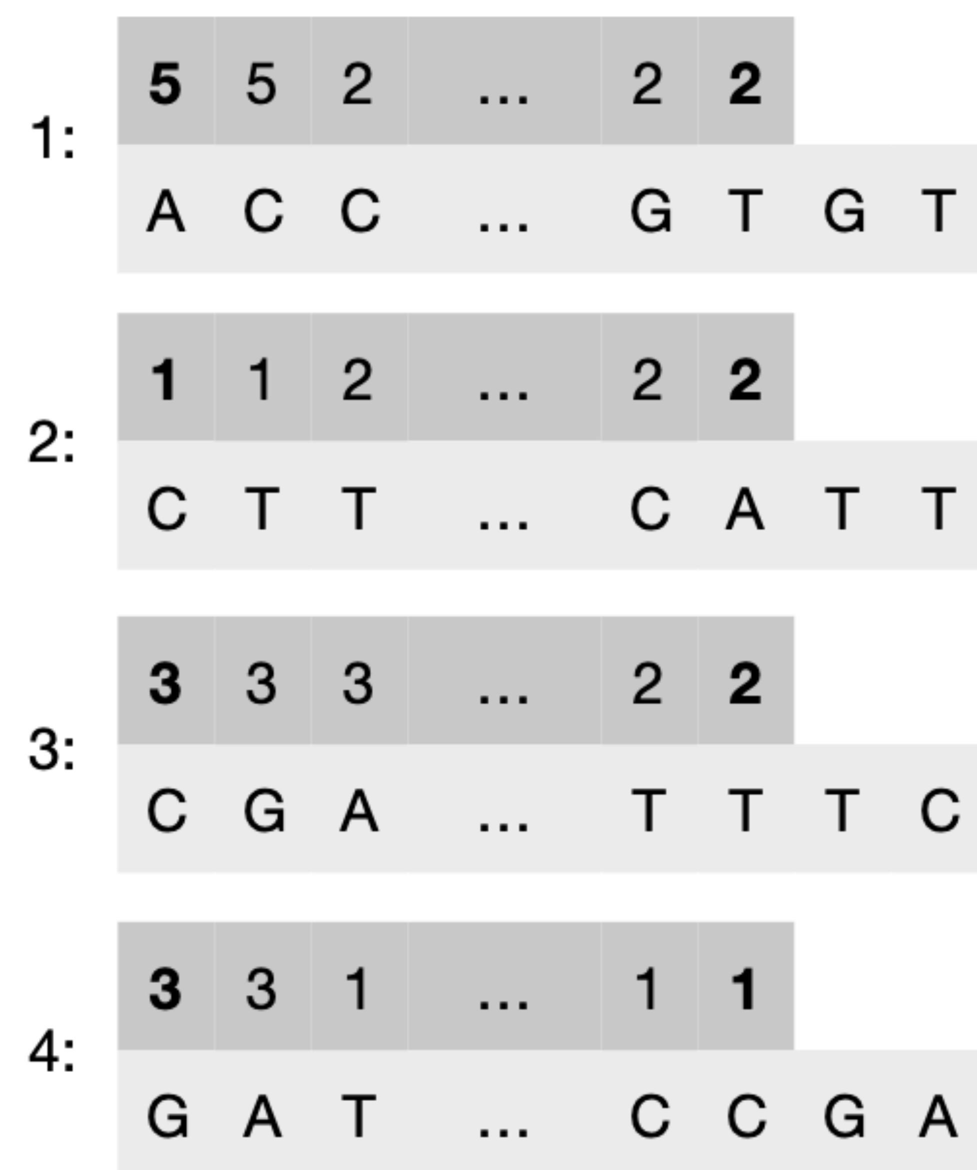
  - The **orientation** of the strings.

- **Note.** Altering $\mathscr{S}$ by changing the order and orientation of the strings **does not** affect the correctness nor the order-preserving property of SSHash.

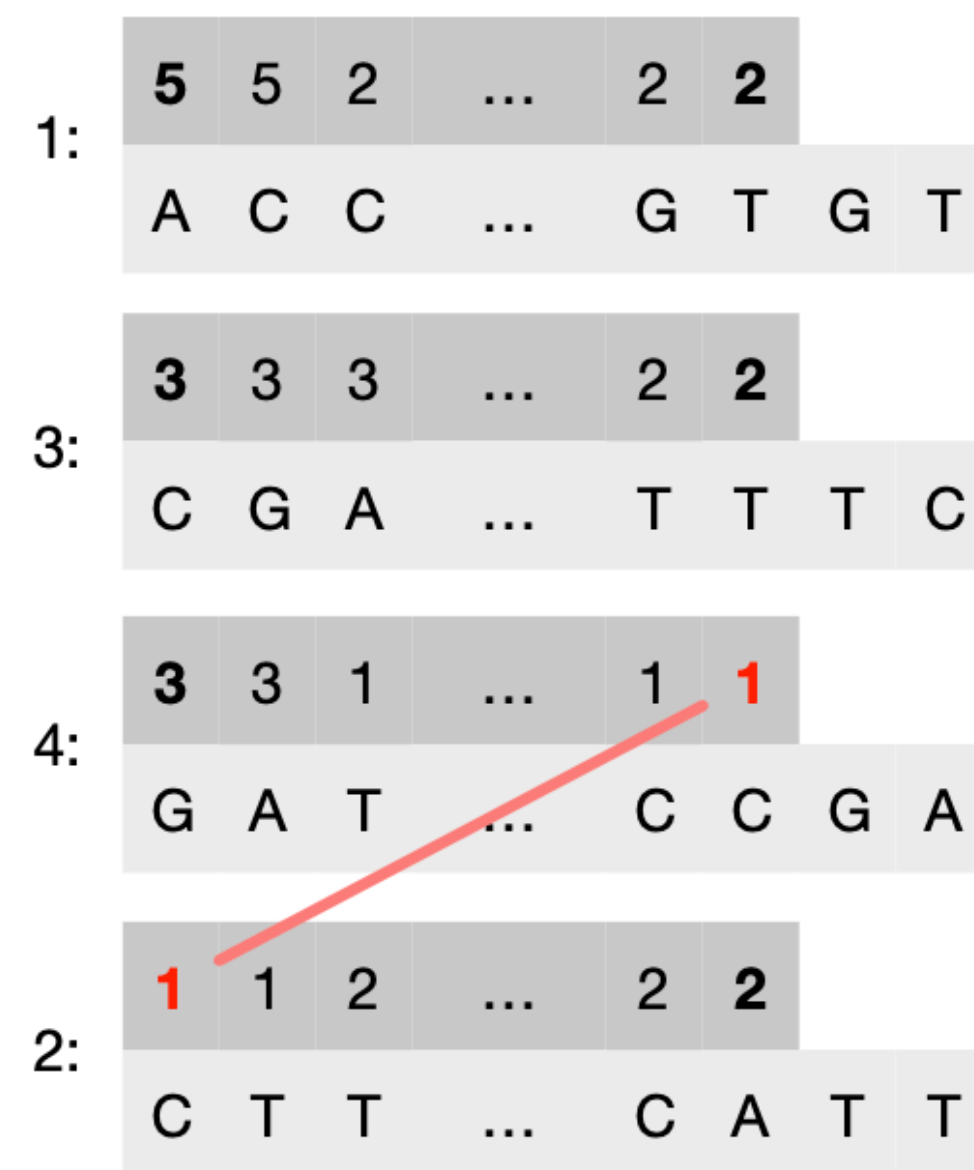
Reducing the Number of Runs

- **Goal.** Compute a *signed* permutation $\pi[1..m]$ where $\pi[i] = j$ indicates that:
  - if $j < 0$: reverse(S_i) has to appear in position $-j$;
  - else: S_i has to appear in position j.

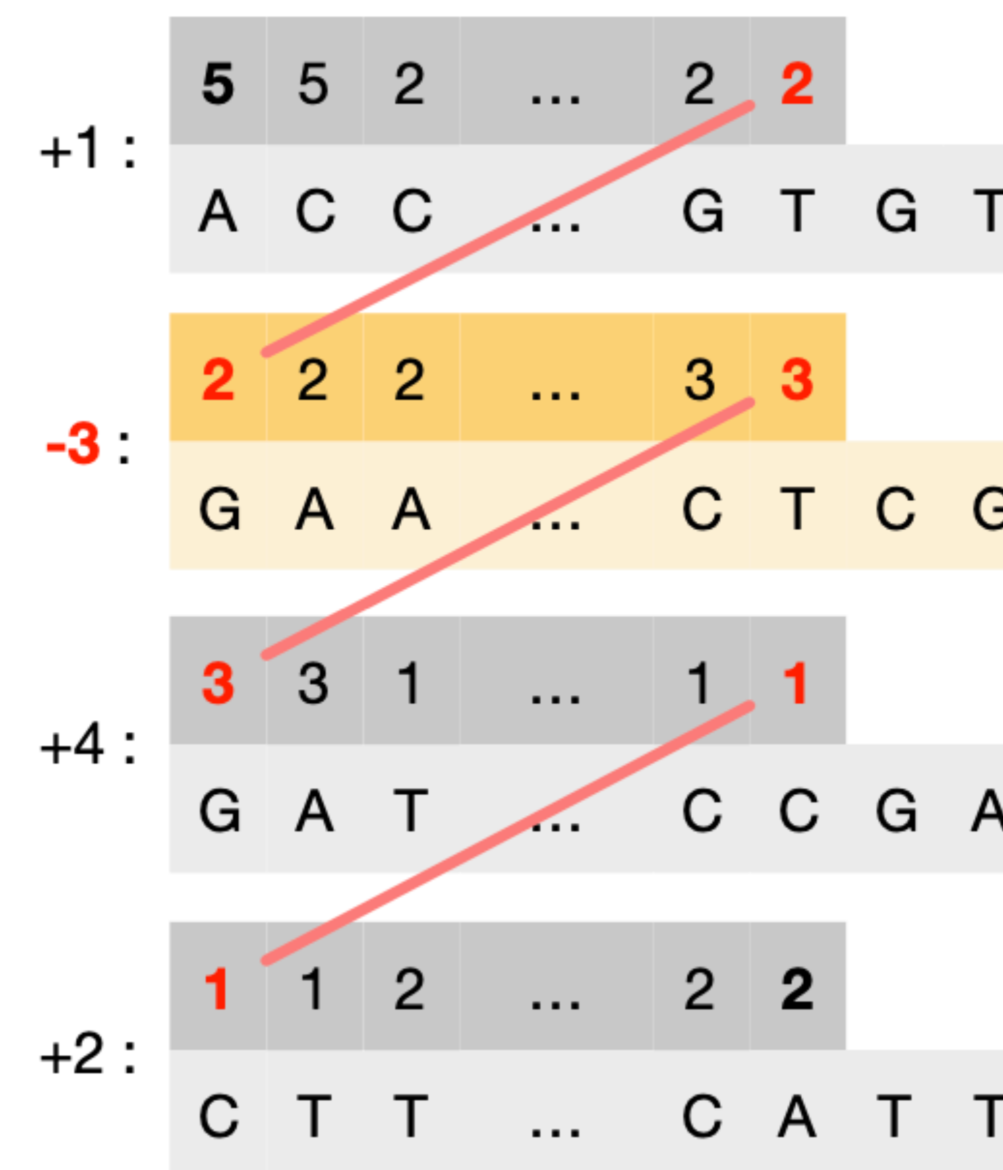
- **Note.** The result π only depends on the **end-point weights** of a string and not on the other weights, nor on the nucleotide sequences.



(a)



(b)



(c)

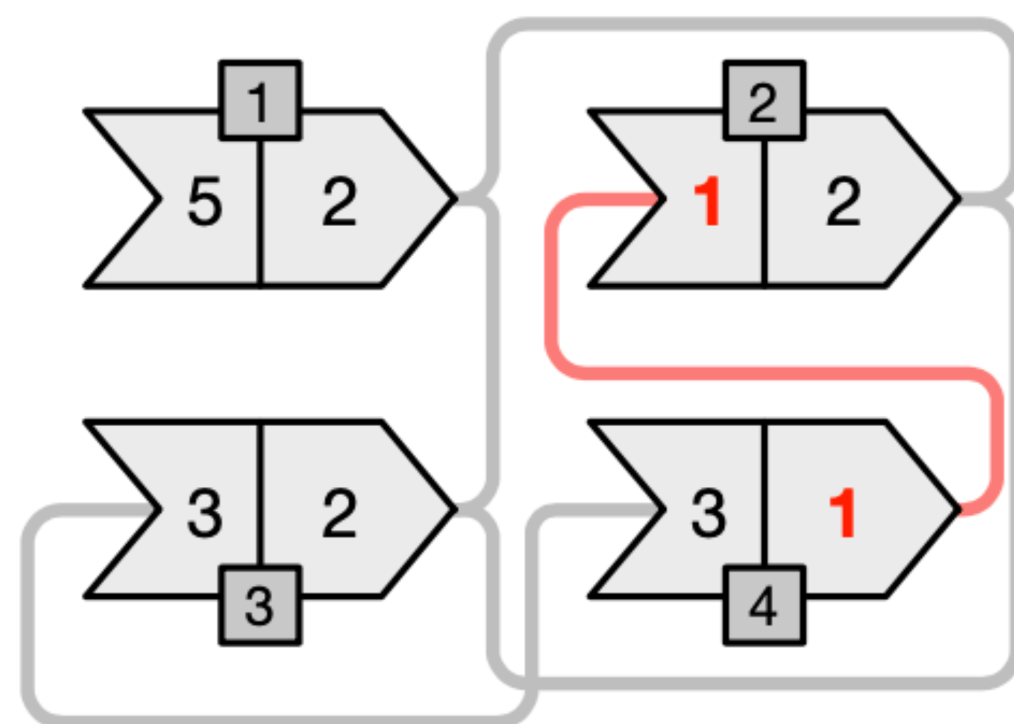
Reversed

$\pi = [+1, +4, -2, +3]$

1 2 3 4

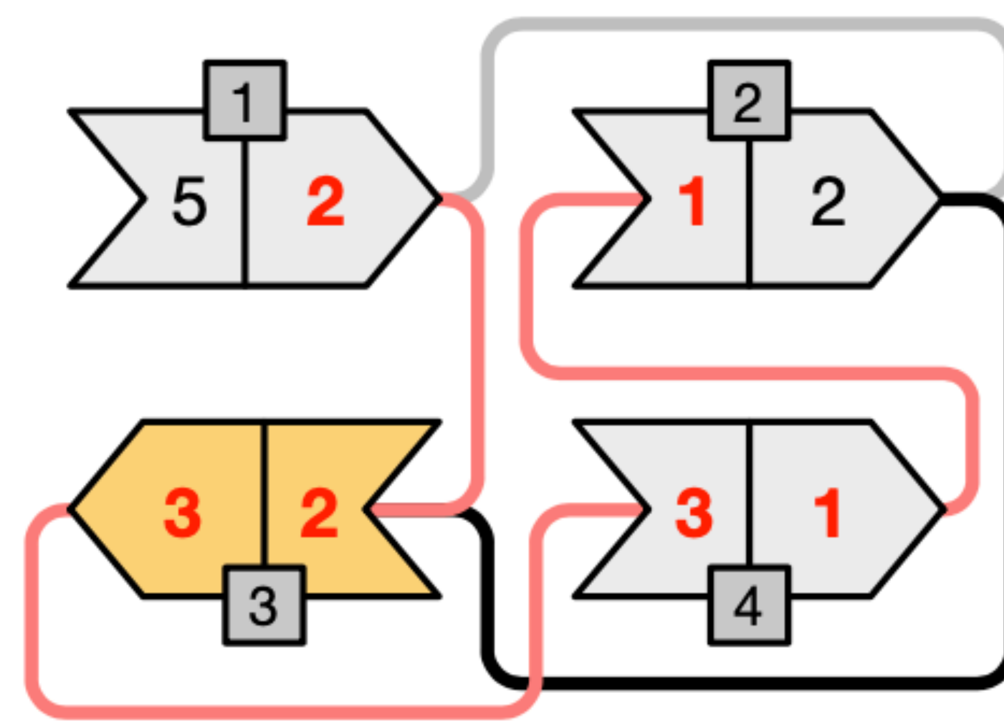
End-Point Weight Graphs and Path Covers

- Since the result π only depends on the end-point weights, it is convenient to consider the **end-point weight graph** $ewG(\mathcal{S})$ for $\mathcal{S}$.
- A (disjoint-node) **path cover** C for $ewG(\mathcal{S})$ determines a signed permutation π.



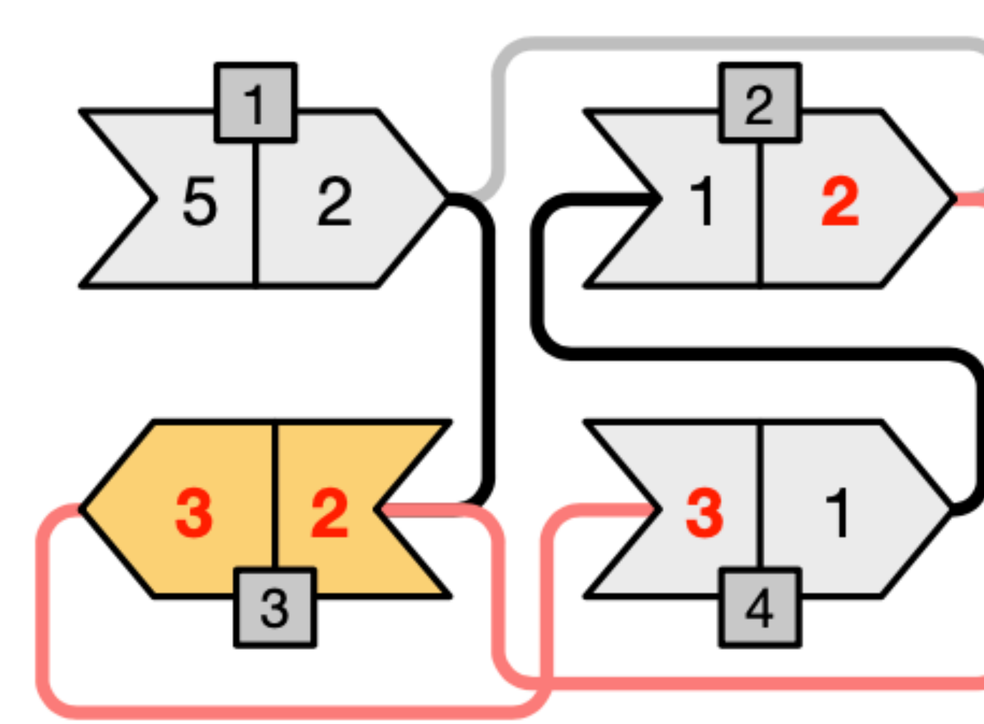
(a)

$C = \{(+4 \rightarrow +2), (+3), (+1)\}$
$\pi = [+1, +4, +2, +3]$



(b)

$C = \{(+1 \rightarrow -3 \rightarrow +4 \rightarrow +2)\}$
$\pi = [+1, +4, -2, +3]$



(c)

$C = \{(+2 \rightarrow -3 \rightarrow +4), (+1)\}$
$\pi = [+4, +1, -2, +3]$

Lower Bound to the Number of Runs

Part 1

- **Q.** Given the strings $S_1, \ldots, S_m$, with S_i having r_i runs in the weights, what is the minimum number of runs we can achieve with our strategy?
  A. We can compute a **lower bound** on the number of runs.
- Let $R = \sum_{i=1}^{m} r_i$. There are at least $R - m$ runs, regardless of the order of the sequences.
- In general, the number of runs will be $R - m + |C|$.
- Every path in C must begin and end with weights that cannot be "glued" with any other path's weights in C, so a new run beings with the first node of every path.
- Therefore, minimizing the number of runs in $\mathcal{S}$ is equivalent to finding a **minimum-cardinality** path cover C for $ewG(\mathcal{S})$.

Lower Bound to the Number of Runs

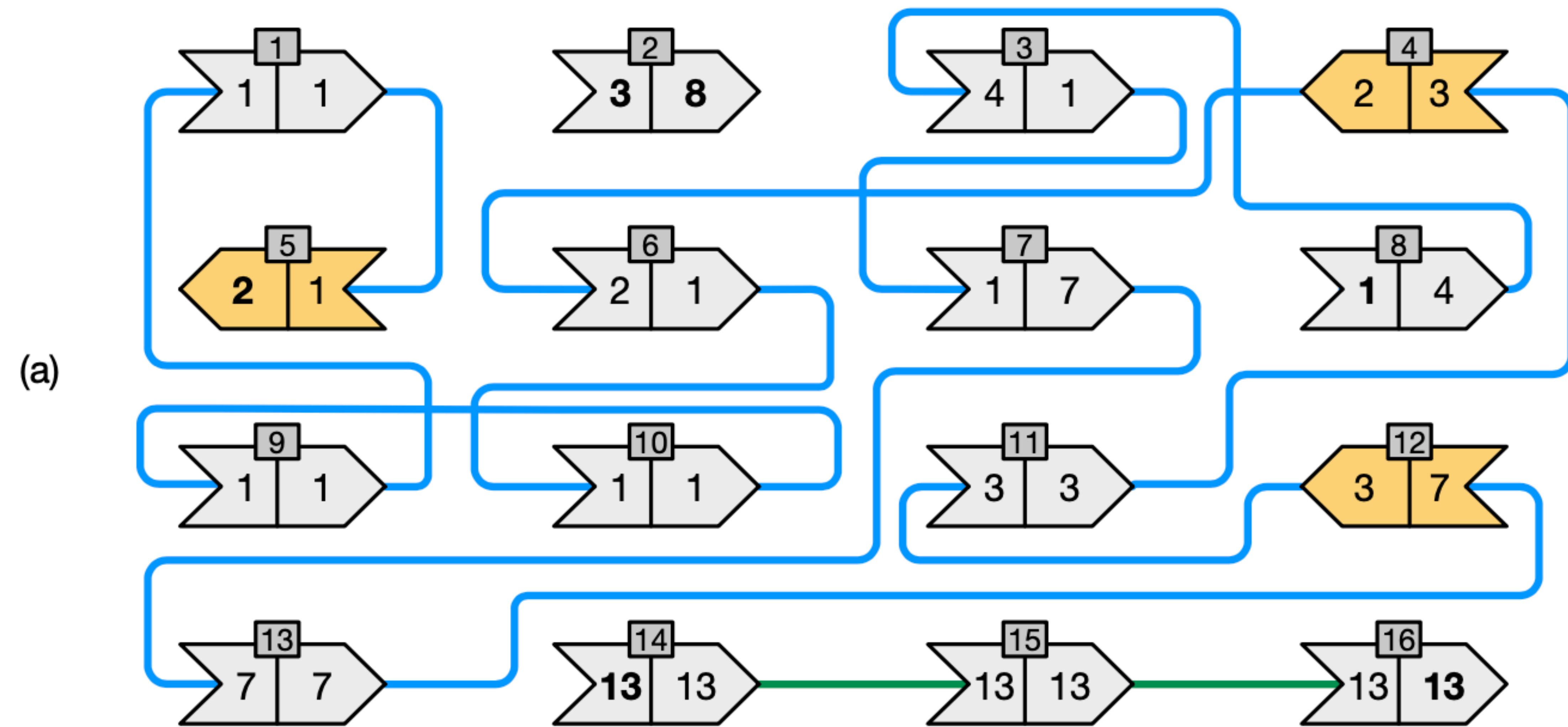
Part 2

- We have $|C| \geq \lceil n_e/2 \rceil$, where n_e is the number of end-point weights that **must necessarily** appear as end-points of the paths in C.
- We derive an expression for n_e that can be computed by just looking at the **number of occurrences** of the end-point weights.

(See the paper for proofs and details.)

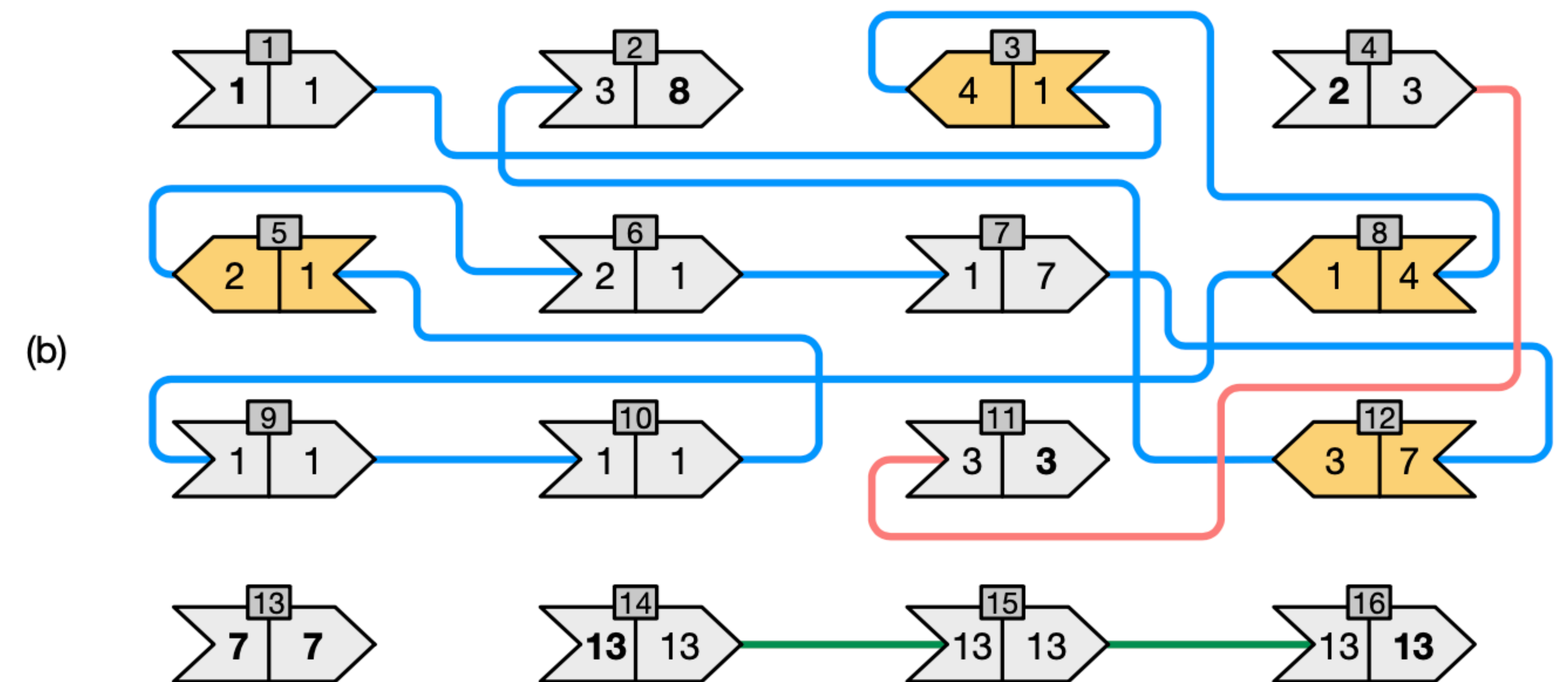
Examples of Path Covers

(a)



C contains **3** paths (**optimal**).

(b)



C contains **4** paths.

Odd:
$f(1) = 11$
$f(2) = 3$
$f(3) = 5$
$f(8) = 1$

Even:
$f(4) = 2$
$f(7) = 4$

Equal:
$f(13) = 6$

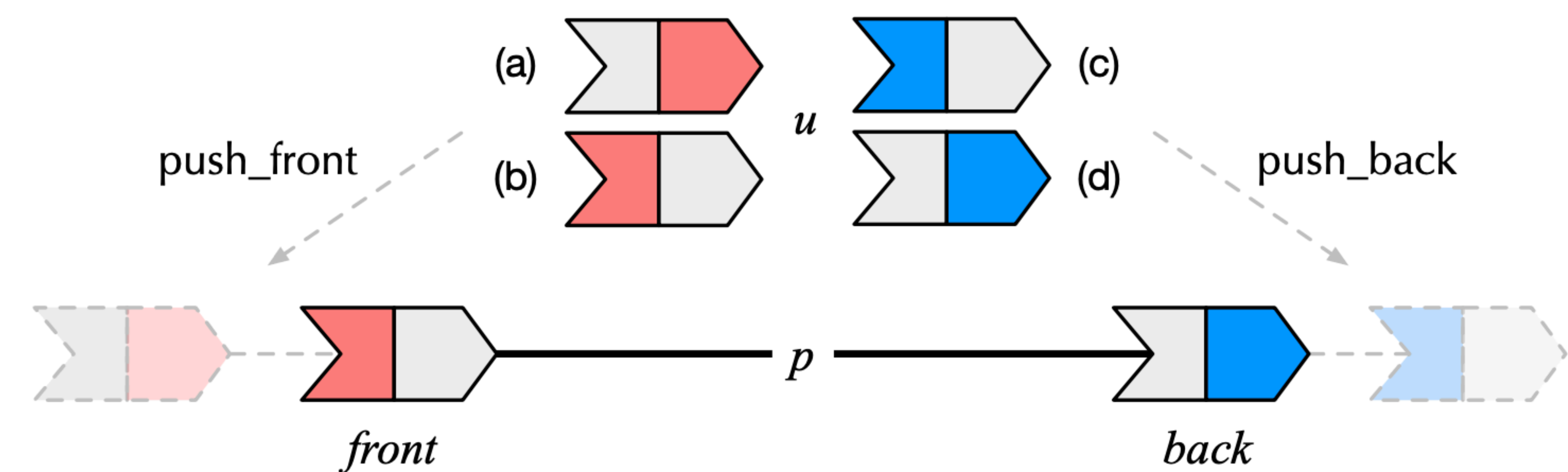
$\rightarrow$ Our lower bound computes $|C| \geq \dfrac{|Odd| + 2|Equal|}{2}$

$= \dfrac{4 + 2 \cdot 1}{2} = 3$ paths.

Greedy Computation of a Path Cover

```
1   cover(ewG(S)):
2       incidence = ∅
3       unvisited = ∅
4       for each node u ∈ ewG(S):
5           unvisited.insert(u)
6           incidence[u.left].insert(u)
7           incidence[u.right].insert(u)
8       while unvisited ≠ ∅ :
9           u = unvisited.take()
10          p = ∅
11          while true :
12              extend p with u
13              unvisited.erase(u)
14              incidence[u.left].erase(u)
15              incidence[u.right].erase(u)
16              if incidence[p.back.right] ≠ ∅ :
17                  u = incidence[p.back.right].take()
18              else if incidence[p.front.left] ≠ ∅ :
19                  u = incidence[p.front.left].take()
20              else : break
21          for each u ∈ p :
22              print (u.sign, u.id)
```

- If we use hash tables to implement *incidence* and *unvisited*, then insert/erase/take are all supported in $O(1)$ on average.
- So the overall complexity (in both time and space) is **linear** in the number of nodes in $ewG(\mathcal{S})$.



Experimental Setup and Datasets

- Processor: Intel(R) Core(TM) i9-9940X CPU @ 3.30GHz
- Compiler and OS: gcc version 11.2.0, Ubuntu 11.2.0-7ubuntu2
- Code in C++17, compiled with flags: `-O3 -march=native`

Some basic statistics for the datasets used in the experiments, for $k = 31$, such as: number of distinct k-mers (n), number of distinct weights ($|\mathcal{D}|$), largest weight (max), expected weight value (E), and empirical entropy of the weights ($H_0(W)$).

Dataset	n	$\lvert\mathcal{D}\rvert$	$\lceil \log_2 \lvert\mathcal{D}\rvert \rceil$	max	$\lceil \log_2 max \rceil$	E	$H_0(W)$
E-Coli	5,235,781	22	5	27	5	1.05	0.206
S-Enterica-100	13,074,614	587	10	3,483	12	37.47	4.420
Human-Chr-13	90,911,778	806	10	6,354	13	1.08	0.160
C-Elegans	94,006,897	398	9	3,478	12	1.07	0.223

Weight Compression

Space for the weights in bits/k-mer, *before* and *after* the run-reduction optimization. In parentheses, we report the compression ratio compared to the empirical entropy.

Dataset	$H_0(W)$	*before*		*after*	
E-Coli	0.206	0.017	(12.11×)	0.014	(15.10×)
S-Enterica-100	4.420	0.592	(7.47×)	0.401	(11.02×)
Human-Chr-13	0.160	0.136	(1.18×)	0.107	(1.50×)
C-Elegans	0.223	0.069	(3.23×)	0.055	(4.05×)

Number of strings (m), number of runs (r) in comparison to the lower bound (r_{lo}), and the run-time of the path cover algorithm (Alg. 4).

Dataset	m	r_{lo}	r		Alg. 4 (ms)	Alg. 4 (ns/node)
E-Coli	2,102	3,723	3,723	(+0.0000%)	0.6	285
S-Enterica-100	150,604	277,649	277,658	(+0.0032%)	53.0	352
Human-Chr-13	266,113	462,175	462,197	(+0.0048%)	94.6	355
C-Elegans	140,452	247,661	247,669	(+0.0032%)	47.1	335

Competitors

- **dBG-FM** [Chikhi et al., 2014]: FM-index [Ferragina and Manzini, 2000]
- **cw-dBG** [Italiano et al., 2021]: *weighted* BOSS [Bowe et al., 2012]
- **BCFS** and **AMB** [Shibuya et al., 2021]: *compressed static functions* (CSFs) — efficient *maps* from k-mers to weights (the k-mers are not represented)

Overall Comparison

Dictionary space in average bits/k-mer and count time in average μsec/k-mer. For reference, we report in gray color the space and time of SSHash *without* the weight information.

Dictionary	E-Coli		S-Enterica-100		Human-Chr-13		C-Elegans	
	space	query-time	space	query-time	space	query-time	space	query-time
dBG-FM, $s = 128$	3.20	14.73	113.78	16.47	3.23	17.40	3.18	18.05
dBG-FM, $s = 64$	4.02	7.91	142.25	11.13	4.07	11.33	4.01	10.89
dBG-FM, $s = 32$	5.65	4.62	198.71	8.57	5.73	8.20	5.67	7.90
cw-dBG, $s = 128$	2.79	109.13	5.59	120.72	2.80	100.88	2.77	127.86
cw-dBG, $s = 64$	2.86	70.93	5.74	85.73	2.86	73.91	2.84	84.19
cw-dBG, $s = 32$	2.99	52.29	6.03	66.25	2.99	59.85	2.97	62.54
SSHash+BCSF	5.07	0.82	11.12	0.89	6.15	1.25	6.00	1.28
SSHash+AMB	4.90	1.34	9.27	1.65	6.08	1.95	5.88	1.97
w-SSHash	4.80	0.37	6.57	0.48	6.04	0.84	5.75	0.85
SSHash	4.79	0.34	6.15	0.41	5.93	0.76	5.69	0.77

Additional Results for w-SSHash

Number of k-mers, number of strings (m), number of runs (r) in comparison to the lower bound (r_{lo}), and the run-time of the path cover algorithm in total seconds (Alg. 4), index space in bits/k-mers (bpk) and total GB, and query time in μsec/k-mer (qtm).

Dataset	n	m	r_{lo}	r		Alg. 4 (sec)	Alg. 4 (ns/node)
Cod	502,465,200	2,406,681	4,183,202	4,183,230	(+0.00067%)	1.2	500
Kestrel	1,150,399,205	682,344	1,140,743	1,140,747	(+0.00035%)	0.3	440
Human	2,505,445,761	13,014,641	22,680,047	22,680,099	(+0.00023%)	7.5	580
Bacterial	5,350,807,438	26,448,286	56,662,230	56,662,304	(+0.00013%)	17.2	650

Dataset	$H_0(W)$	bpk		GB	qtm
Cod	0.441	6.98+0.19	(2.35×)	0.45	1.3
Kestrel	0.089	6.49+0.02	(3.80×)	0.94	1.1
Human	0.453	8.28+0.22	(2.06×)	2.66	1.6
Bacterial	1.890	8.22+0.24	(7.81×)	5.66	1.9

Conclusions

- SSHash is an efficient solution to the *weighted k-mer dictionary problem*: good trade-off between space and time.

- Algorithmic tool-box:
  - SPSS, minimizers, MPHF (not covered in this talk, see talk@ISMB-2022)
  - Elias-Fano, RLE.

- Order of the k-mers induces **runs** in the weights: suitable for RLE.

- Permuting and accordingly orienting the strings in the SPSS reduces the number of runs in the weights essentially to the minimum according to the lower bound.

- Compared to BWT-based indexes: one order of magnitude faster for “just” 2X more space. Compared to other hashing schemes: 2-5X smaller with comparable of faster query time.

- Weights add very small extra space and do not impact query time.

Thank you for the attention!