Shark 🦈
Fishing in a sample to discard irrelevant RNA-Seq reads

Paola Bonizzoni, Tamara Ceccato, Gianluca Della Vedova,
Luca Denti, **Yuri Pirola**, Marco Previtali and Raffaella Rizzi

DISCo - Univ. degli Studi di Milano-Bicocca

DSB 2020 / Rennes / Feb. 4+5, 2020

The problem

- Sequencing technologies produce a lot of data
- Sequencing datasets are piling up in public repositories
- What if I want to analyze only a subset of genes (RNA-Seq studies)?
- **Notice:** aligning RNA-Seq reads to the genome is **not** an easy task!

Outline

- The *Gene Assignment Problem*
- The Algorithm
- Experimental results:
  - Synthetic dataset
  - Reproduction of real-world analyses

The *Gene Assignment Problem*

Given:

- S a set of RNA-Seq reads
- G a set of genes (genomic sequences)
- two parameters $k \in \mathbb{N}^+$ and $\tau \in [0, 1]$

Compute $\{S_1, \ldots, S_{|G|}\}$ s.t. for each $s \in S_i \subseteq S$:

- the set of bases of s "covered" by at least a k-mer shared with g_i has cardinality at least $\tau \cdot |s|$
- such a cardinality is maximum (w.r.t. other $g_j \in G$).

The *Gene Assignment Problem* - Goals

We want to be:

- alignment-free (reduce potential alignment biases)
- highly sensitive (almost no reads are lost)
- as specific as possible (the dataset is reduced as much as possible)
- very fast
- use a modest amount of memory

The algorithm

High-level idea:

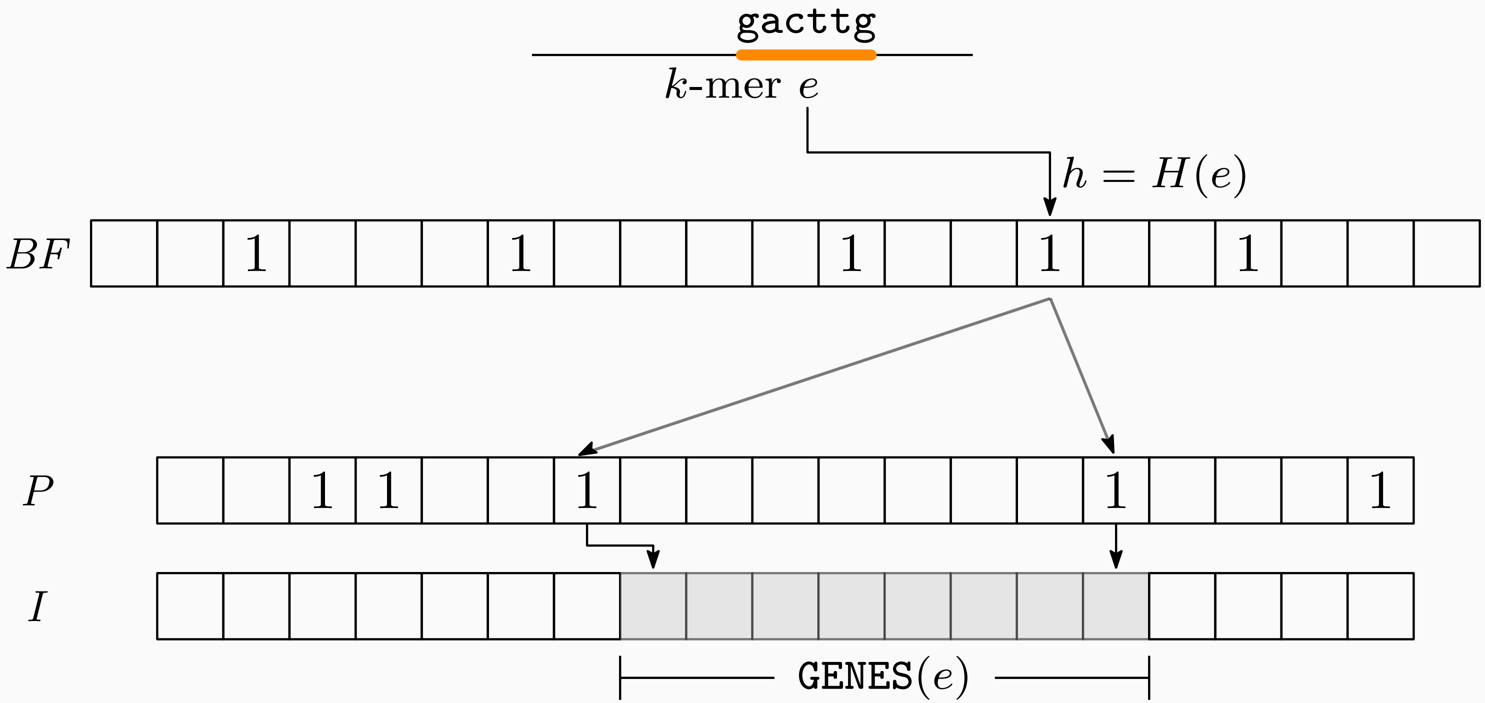
1. Index the $\boldsymbol{k}$-mers of gene sequences in $\boldsymbol{G}$
2. Assign each read $\boldsymbol{s} \in \boldsymbol{S}$ to its set of genes (if any)

The algorithm - Indexing genes

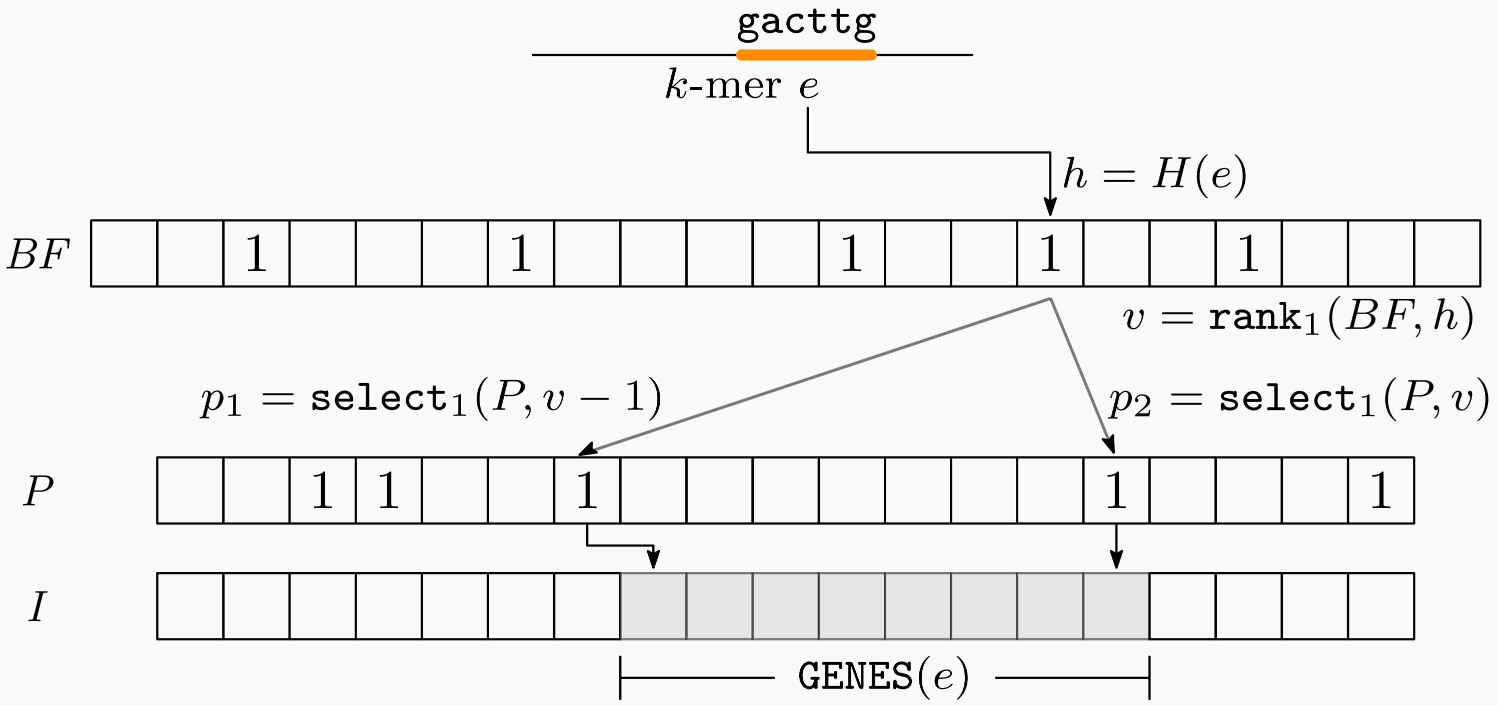
Index $\langle BF, I, P \rangle$**:**

- a Bloom filter BF storing k-mers of G (with a single hash function H)
- an integer vector I storing indices of genes associated to each k-mer
- a bit-vector P providing a mapping between BF and I by "tagging" the boundaries among the different subsets of genes in I

The algorithm - Indexing genes



The algorithm - Indexing genes


gacttg
k-mer e
$h = H(e)$
BF
1 1 1 1 1
$v = \texttt{rank}_1(BF, h)$
$p_1 = \texttt{select}_1(P, v-1)$
$p_2 = \texttt{select}_1(P, v)$
P
1 1 1 1 1
I
$\texttt{GENES}(e)$

The algorithm - How to build the index?

1. Scan each k-mer of the gene sequences in G and store them in BF (using a single hash function H)
2. Associate an empty list L_v to each `1` in BF
3. Re-scan each k-mer e of each $g_i \in G$ and add i to the list L_v associated to the `1` at position $H(e)$ in BF
4. Copy (preserving the order) all the lists L_v into I while tagging the boundaries between lists with a `1` in P

The algorithm - Assigning reads

To assign each read $\boldsymbol{s} \in \boldsymbol{S}$

1. For each $\boldsymbol{k}$-mer $\boldsymbol{e} \in \boldsymbol{s}$:
    1. Query the index to find the genes containing $\boldsymbol{e}$ (FP are possible due to $\boldsymbol{BF}$)
    2. For each gene, compute how many bases are covered by $\boldsymbol{e}$ but not by previous $\boldsymbol{k}$-mers
2. Output the IDs of genes that cover the largest number of bases of $\boldsymbol{s}$ *if* $\geq \boldsymbol{\tau} \cdot |\boldsymbol{s}|$ *bases*

Implementation

- `Shark` is available at

  https://github.com/AlgoLab/shark

- Binaries through Bioconda

```
conda config --add channels bioconda
conda install shark
```

- Libraries:
  - `sdsl-lite` for DS
  - Intel TBB for multi-threading

Does it work?

Experimental results

- How $\boldsymbol{k}$ and $\boldsymbol{\tau}$ affect sensitivity and specificity?

  $\rightarrow$ synthetic datasets

- Is `Shark` useful?
  - Does it change the results?
  - Does it make analyses faster and/or less memory hungry?

  $\rightarrow$ replication of "real-world" analyses

Synthetic datasets - Data

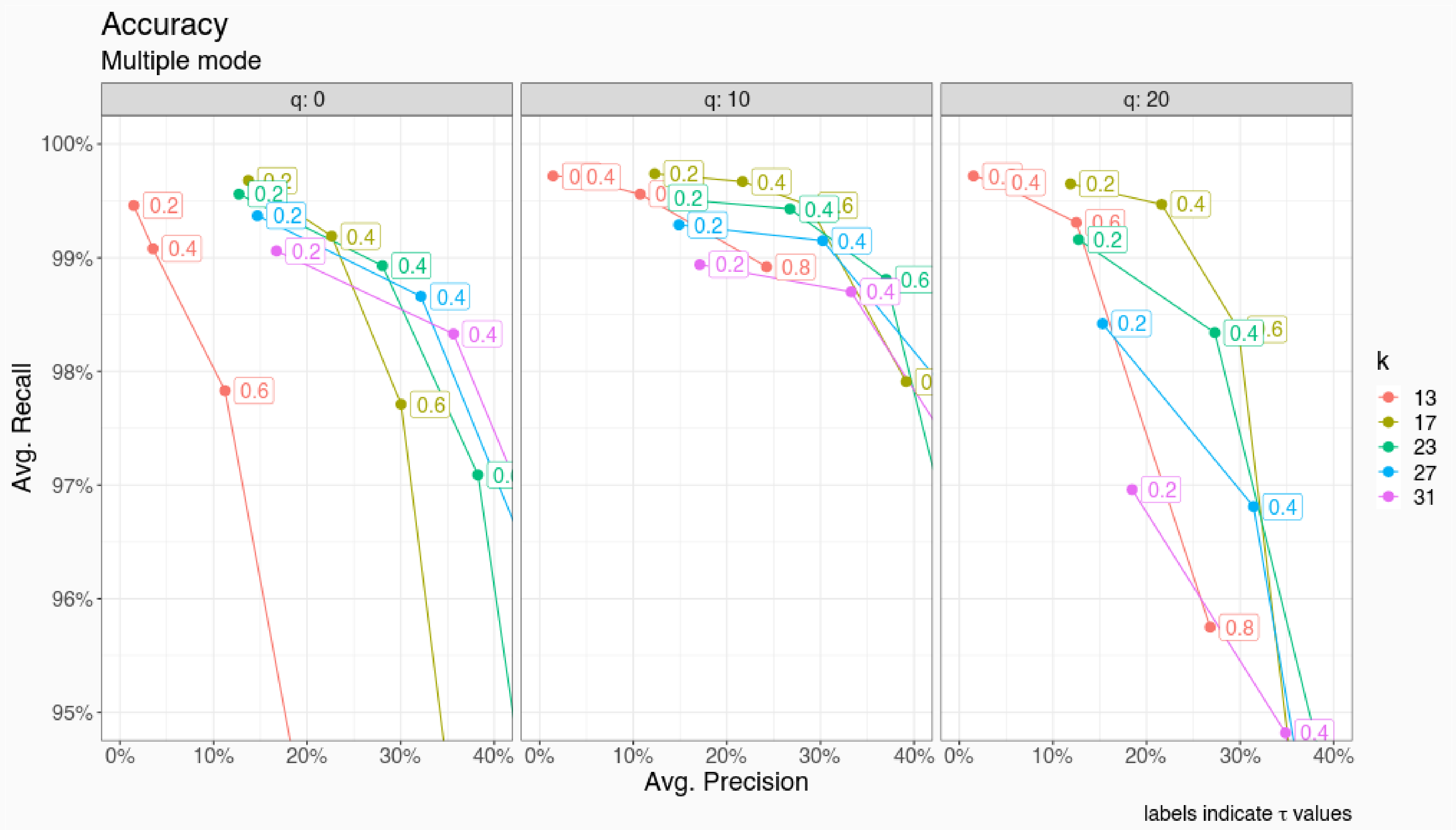
Input data:

- RNA-Seq sample of 10M 100bp-long reads on chr1, 17, 21
- $k \in \{13, 17, 23, 27, 31\}$
- $\tau \in \{0.2, 0.4, 0.6, 0.8\}$
- k-mers covering bases with PHRED quality score $< q$ have been discarded, with $q \in \{0, 10, 20\}$ ($q = 0$ means no filter)
- 10 different instances selecting random subsets of 100 genes each

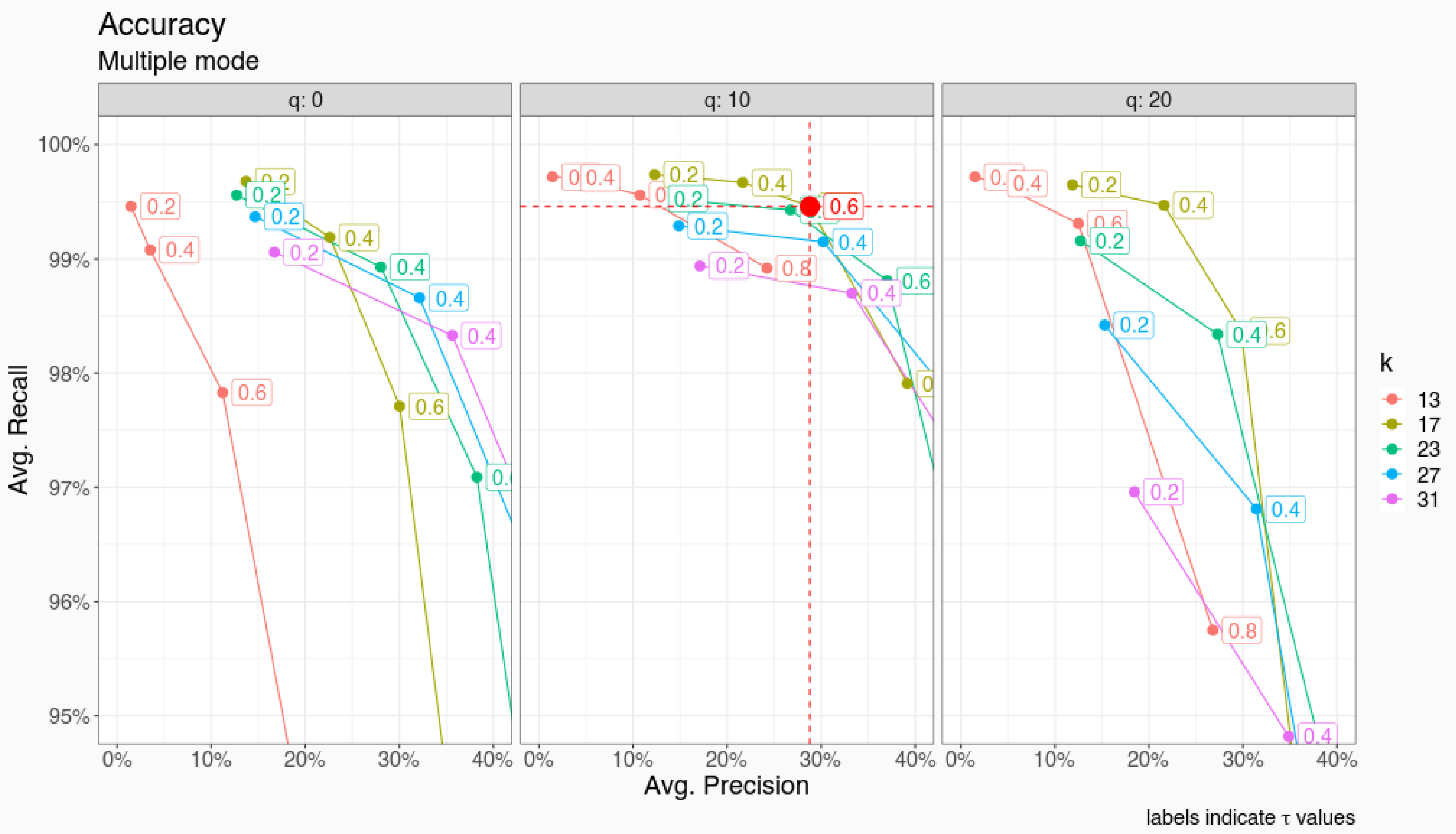
Accuracy measures:

- $\text{Recall} = \text{TP}/(\text{TP} + \text{FN})$
- $\text{Precision} = \text{TP}/(\text{TP} + \text{FP})$

Synthetic datasets - Results

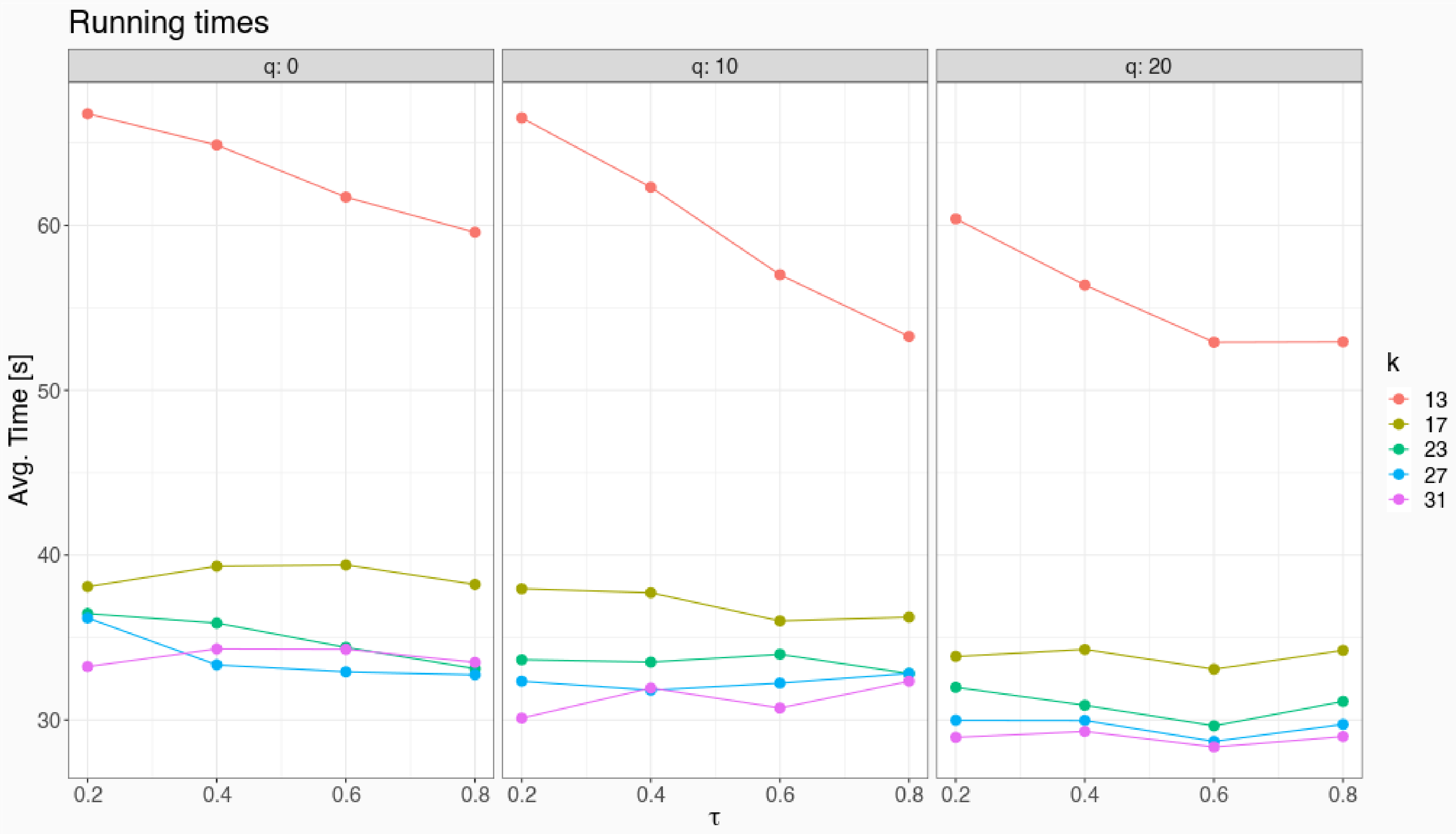


Synthetic datasets - Results



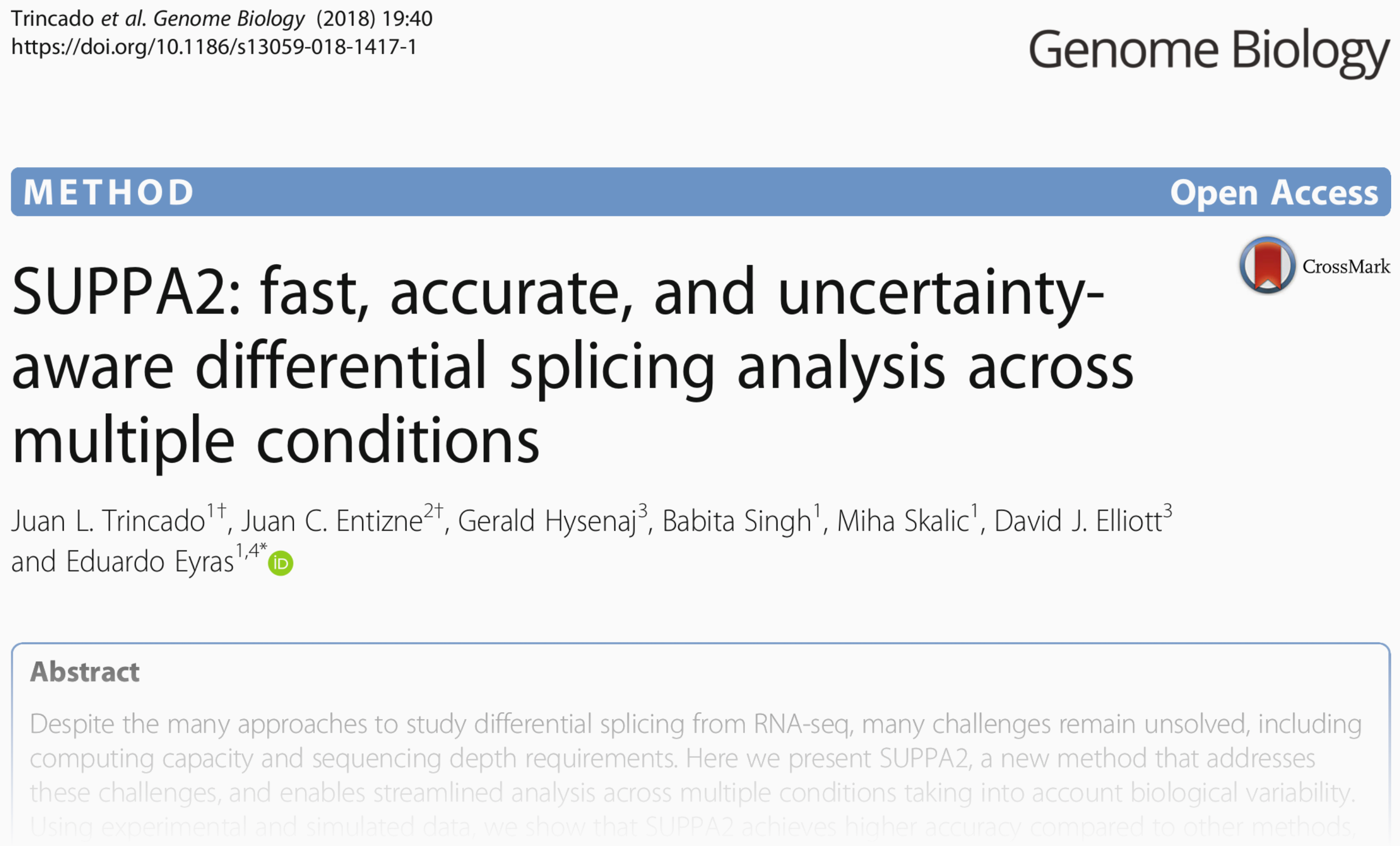
Good compromise: $k = 17, \tau = 0.6, q = 10 \rightarrow R = 99.46\%, P = 28.8\%$.

Synthetic datasets - Results



Memory usage was always below 2.1GB

Replication of real-world analyses



Trincado *et al. Genome Biology* (2018) 19:40
https://doi.org/10.1186/s13059-018-1417-1

Genome Biology

METHOD **Open Access**

CrossMark

SUPPA2: fast, accurate, and uncertainty-aware differential splicing analysis across multiple conditions

Juan L. Trincado[1†], Juan C. Entizne[2†], Gerald Hysenaj[3], Babita Singh[1], Miha Skalic[1], David J. Elliott[3] and Eduardo Eyras[1,4*]

Abstract

Despite the many approaches to study differential splicing from RNA-seq, many challenges remain unsolved, including computing capacity and sequencing depth requirements. Here we present SUPPA2, a new method that addresses these challenges, and enables streamlined analysis across multiple conditions taking into account biological variability. Using experimental and simulated data, we show that SUPPA2 achieves higher accuracy compared to other methods,

Replication of real-world analyses

Aim: Differential analysis of AS events

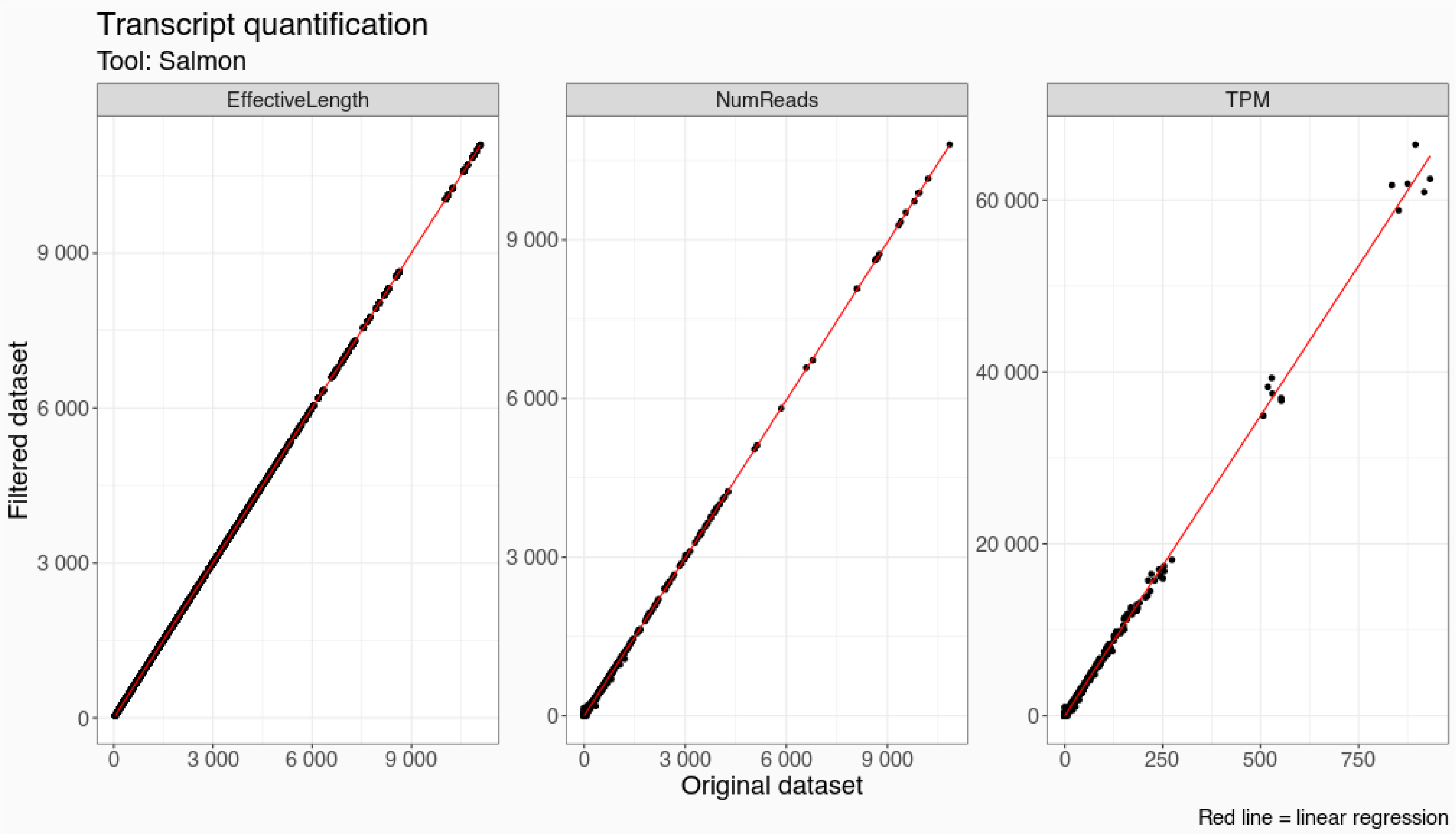
Input data:

- 6 PE RNA-Seq samples (~180M 101bp-long reads)
- 82 distinct genes with 83 exp. validated exon skipping events

Pipelines:

- SplAdder
- rMATS
- SUPPA2

Replication - Transcript quantification

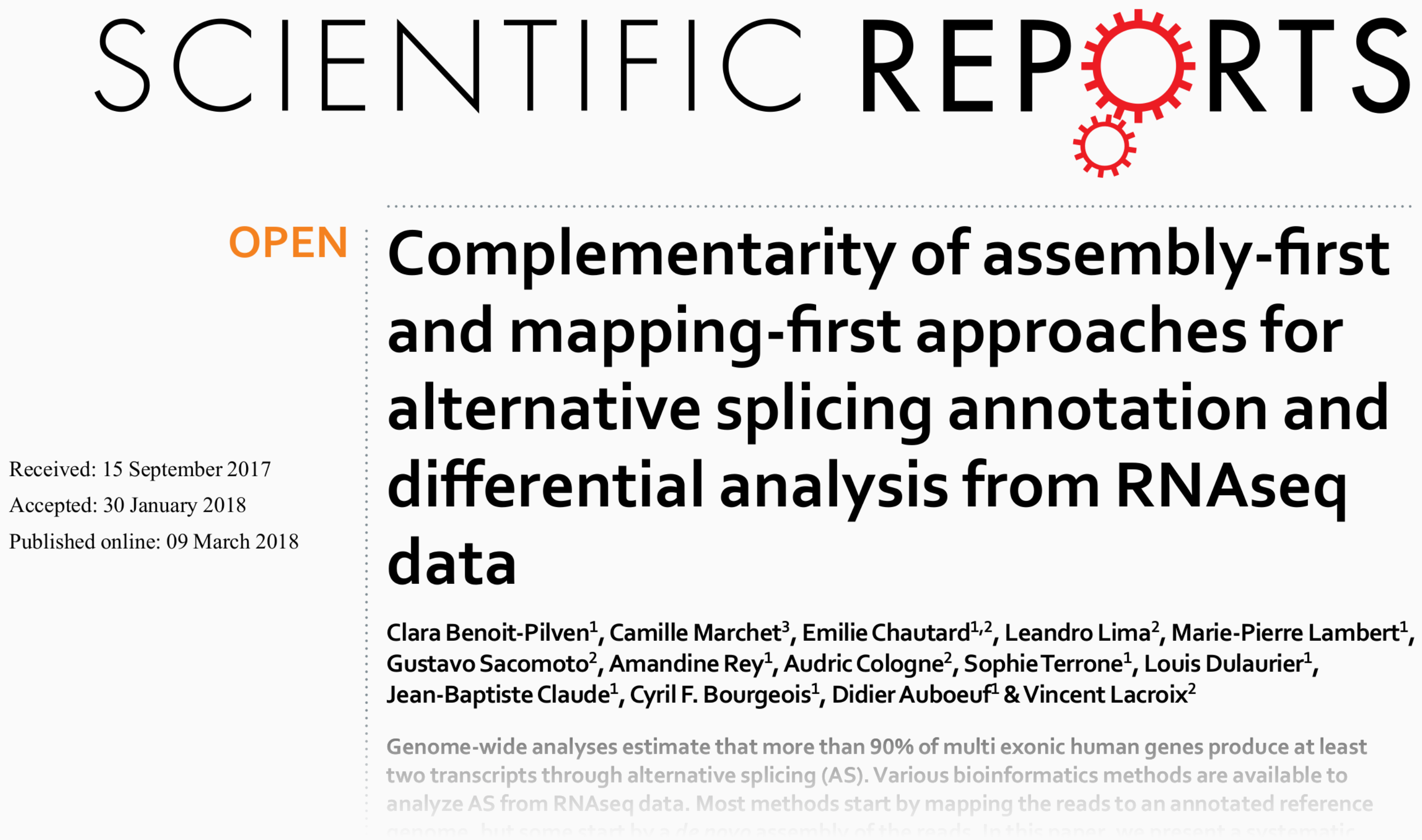


Replication - Diff. expressed events

Pipeline	Validated events		Time [min]	Memory [GB]
	All	p-value < 0.05		
rMATS	78	63	308	33.9
Shark + rMATS	78	63	142	33.9
SplAdder	56	NA	796	33.9
Shark + SplAdder	56	NA	295	33.9
SUPPA2	66	37	65	4.3
Shark + SUPPA2	66	43	34	4.4

Thanks to `Shark` we can also decrease the max memory consumption to less than 16GB w/o affecting running times (data not shown).

Speeding-up assembly-first analyses



SCIENTIFIC REPORTS

OPEN

Received: 15 September 2017
Accepted: 30 January 2018
Published online: 09 March 2018

Complementarity of assembly-first and mapping-first approaches for alternative splicing annotation and differential analysis from RNAseq data

Clara Benoit-Pilven[1], Camille Marchet[3], Emilie Chautard[1,2], Leandro Lima[2], Marie-Pierre Lambert[1], Gustavo Sacomoto[2], Amandine Rey[1], Audric Cologne[2], Sophie Terrone[1], Louis Dulaurier[1], Jean-Baptiste Claude[1], Cyril F. Bourgeois[1], Didier Auboeuf[1] & Vincent Lacroix[2]

Genome-wide analyses estimate that more than 90% of multi exonic human genes produce at least two transcripts through alternative splicing (AS). Various bioinformatics methods are available to analyze AS from RNAseq data. Most methods start by mapping the reads to an annotated reference genome, but some start by a *de novo* assembly of the reads. In this paper, we present a systematic

Speeding-up assembly-first analyses

Preliminary results:

- Almost no changes in stat. signif. results
- Significant speed-up (from ~25h to ~3h)
- Significantly less memory (from ~12.5GB to ~5.5GB)

Ongoing work:

- Manual inspection of results

Conclusions

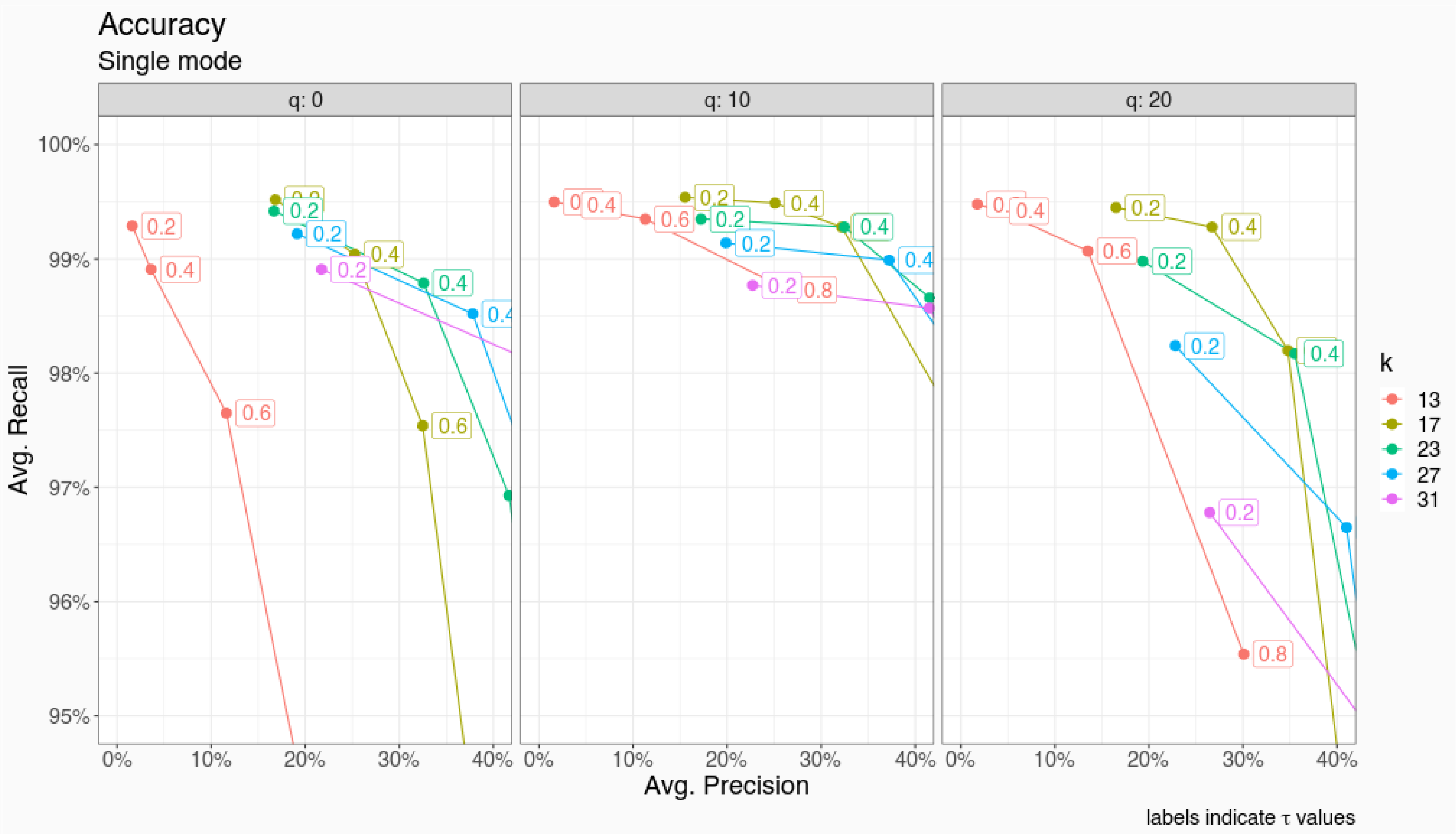
- `Shark` speeds up analyses by focusing on reads likely sequenced from genes of interest
    - Highly sensitive (almost no reads are lost)
    - Good precision (dataset is substantially reduced)
- For more details → preprint on bioRχiv
- **Ongoing work:**
    - Algorithmic solution is simple (but effective).

      Can we do better?
    - Clearly not suitable for all kind of analyses (i.e., transcriptome-wide analyses).

      But, if we want to focus on specific genes, do results change?

Thanks!

Synthetic datasets - Results


Accuracy
Single mode
q: 0
q: 10
q: 20
Avg. Recall
Avg. Precision
k
labels indicate τ values