Large Scale Sequence Search

Fast search of thousands of short-read sequencing experiments.

SBT introduced by **Solomon and Kingsford.**

Nature biotechnology. 2016 doi: 10.1038/nbt.3442

Problem:

The vast repository of publicly-available data (e.g., the SRA) is essentially unsearchable by sequence. Current solutions (BLAST, STAR) too slow. What if I find a novel txp and want to search the SRA for it?

Solution:

A hierarchical index of k-mer content represented approximately via Bloom filters. Returns “yes/no” results for individual experiments -> “yes” results can be searched using more traditional methods.

Recall the bloom filter

Bloom Filters

Originally designed to answer *probabilistic* membership queries:

Is element e in my set S?

If yes, **always** say yes

If no, say no **with large probability**

False positives can happen; false negatives cannot.

Bloom Filters

For a set of size N, store an array of M bits

Use k different hash functions, $\{h_0, \ldots, h_{k-1}\}$

To insert e, set $A[h_i(e)] = 1$ for $0 < i < k$

To query for e, check if $A[h_i(e)] = 1$ for $0 < i < k$

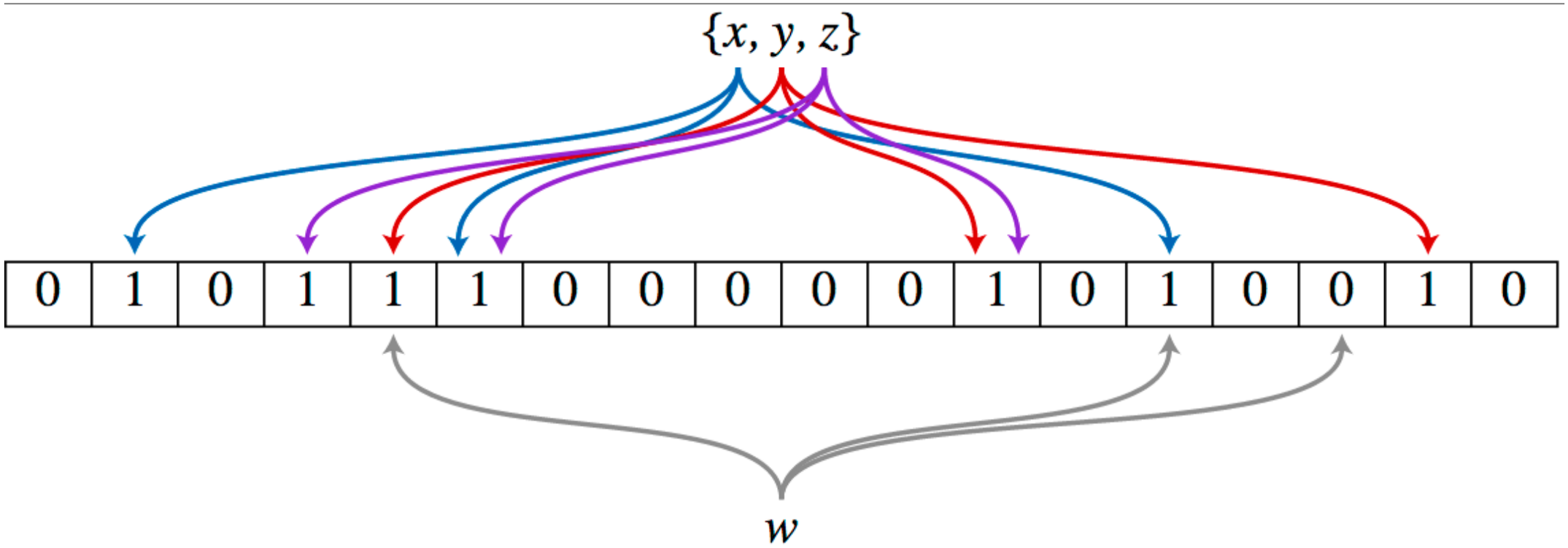


Image by David Eppstein - self-made, originally for a talk at WADS 2007

Bloom Filters

If hash functions are good and sufficiently independent, then the probability of false positives is low and controllable.

How low?

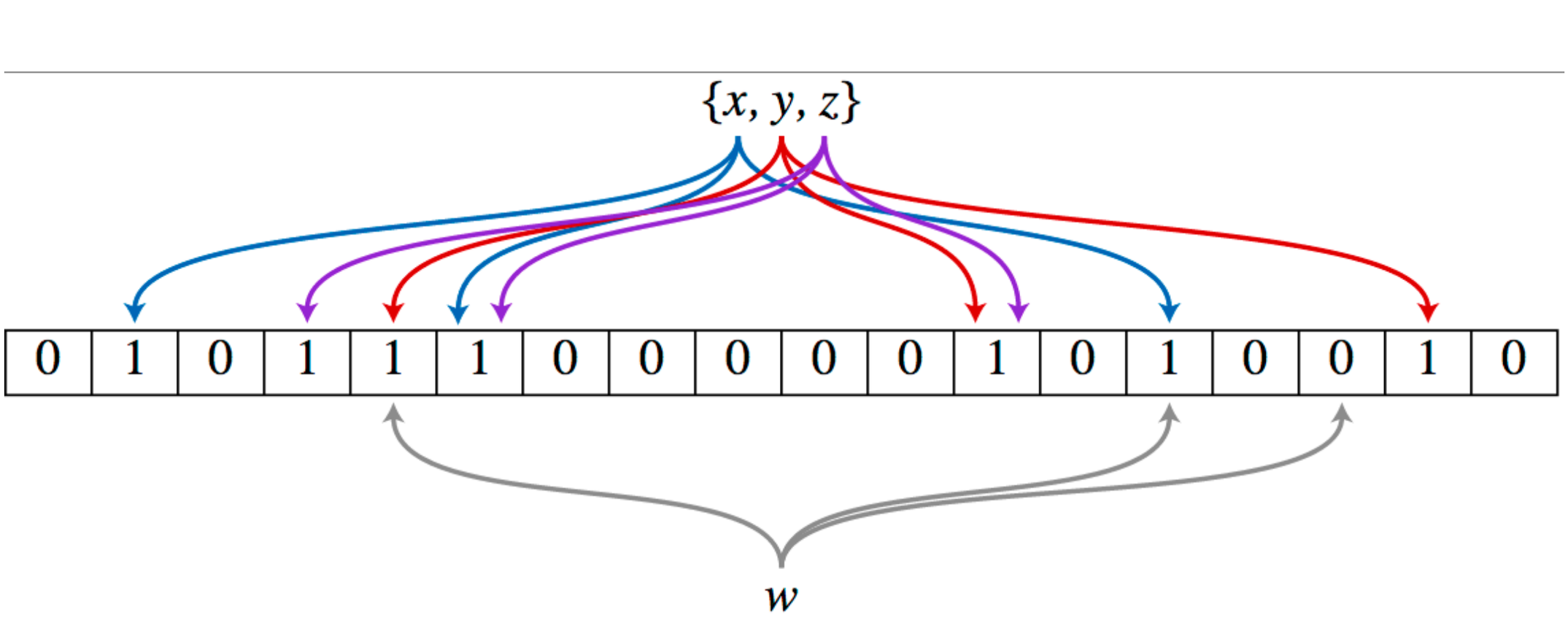


Image by David Eppstein - self-made, originally for a talk at WADS 2007

False Positives

Let q be the fraction of the m-bits which remain as 0 after n insertions.

The probability that a randomly chosen bit is 1 is 1-q.

But we need a 1 in the position returned by k different hash functions; the probability of this is $(1-q)^k$

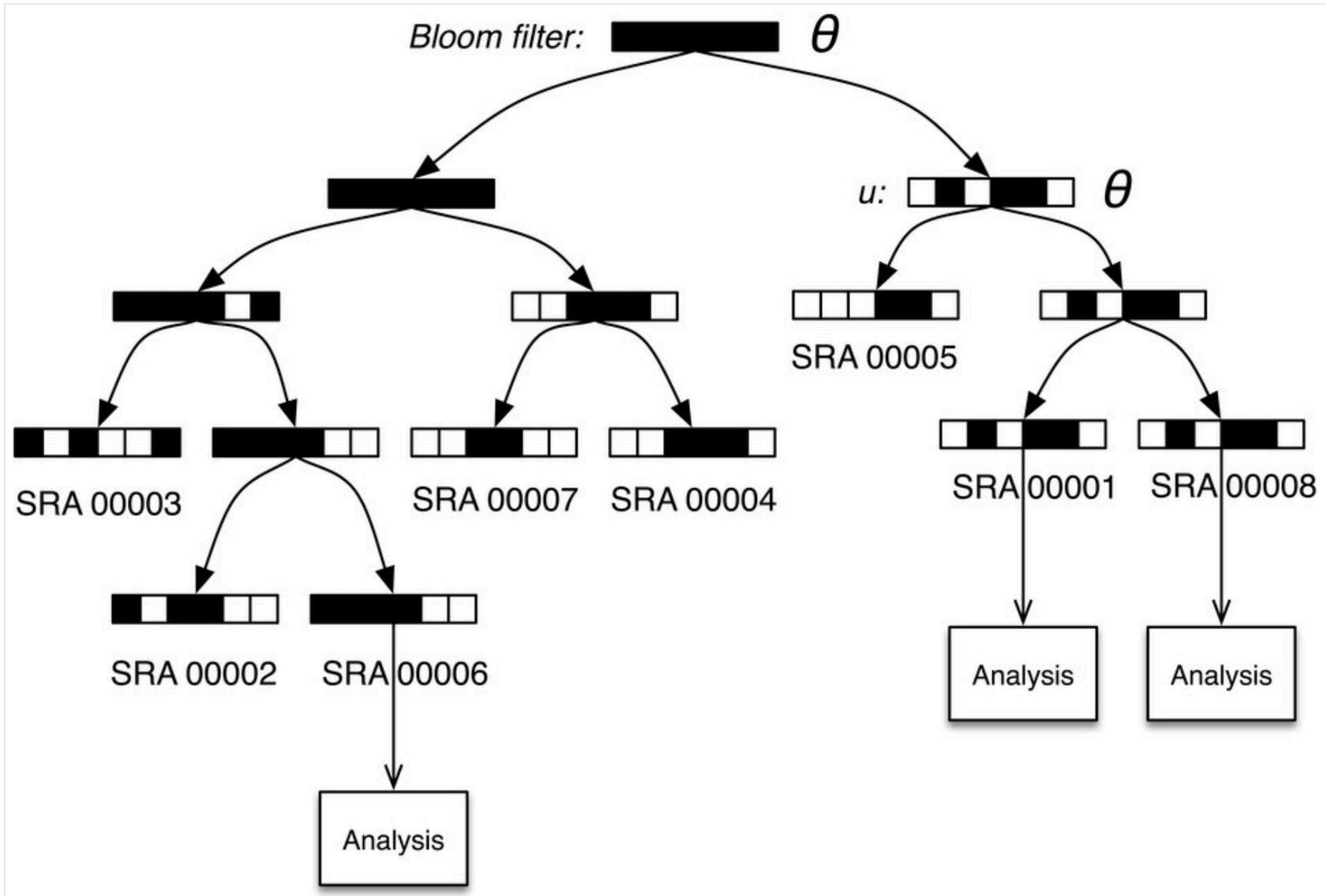
We can derive a formula for the expected value of q, for a filter of m bits, after n insertions with k different hash functions:

$$E[q] = (1 - 1/m)^{kn}$$

*analysis of Mitzenmacher and Upfal

SBT

An SBT is a binary tree of bloom filters, where leaves represent the k-mer set of a single *sample.*



Each node contains a bloom filter that holds the kmers present in the sequencing experiments under it. θ is the fraction of kmers required to be found at each node in order to continue to search its subtree. The SBT returns the experiments that likely contain the query sequence on which further analysis can be performed.

Solomon, Brad, and Carl Kingsford. "Fast search of thousands of short-read sequencing experiments." *Nature biotechnology* 34.3 (2016): 300-302.

SBT Operations

Construction (repeatedly insert samples *s*):

Let b(*s*) be the bloom filter of sample *s*

Use b(*s*) to walk from the root of T to the leaves

For a node *u*:

If *u* has a single child, insert b(s) as the other child

If *u* has 2 children recurse into child with < hamming dist to b(s)

If *u* is a leaf (an experiment), create a parent with filter b(u) U b(s)

SBT Operations

Query (given collection of k-mers K_q, parameter θ):

For a node *u*:

Hash elements of K_q and check if at least θ | K_q| k-mers exist

If not then this sub-tree cannot θ-match our query

Else continue searching both children recursively

The implementation allows each query k-mer to be given a "weight" or importance.

Note: because we are searching a set rather than a single element, we can usually accommodate a *much* larger false positive rate (they use 0.5).

Theorem 2

Let q be a query string containing ℓ distinct k-mers. If we treat the k-mers of q as being independent, the probability that $>\lfloor\theta\ell\rfloor$ false-positive k-mers appear in a filter U with FPR ξ is

$$1-\sum_{i=0}^{\lfloor\theta\ell\rfloor}\binom{\ell}{i}\xi^{i}(1-\xi)^{\ell-i} \qquad (4)$$

The above expression is nearly 0 when $\xi \ll \theta$.

SBT Tricks

High false positive rate lets filters be small (& use only a single hash)

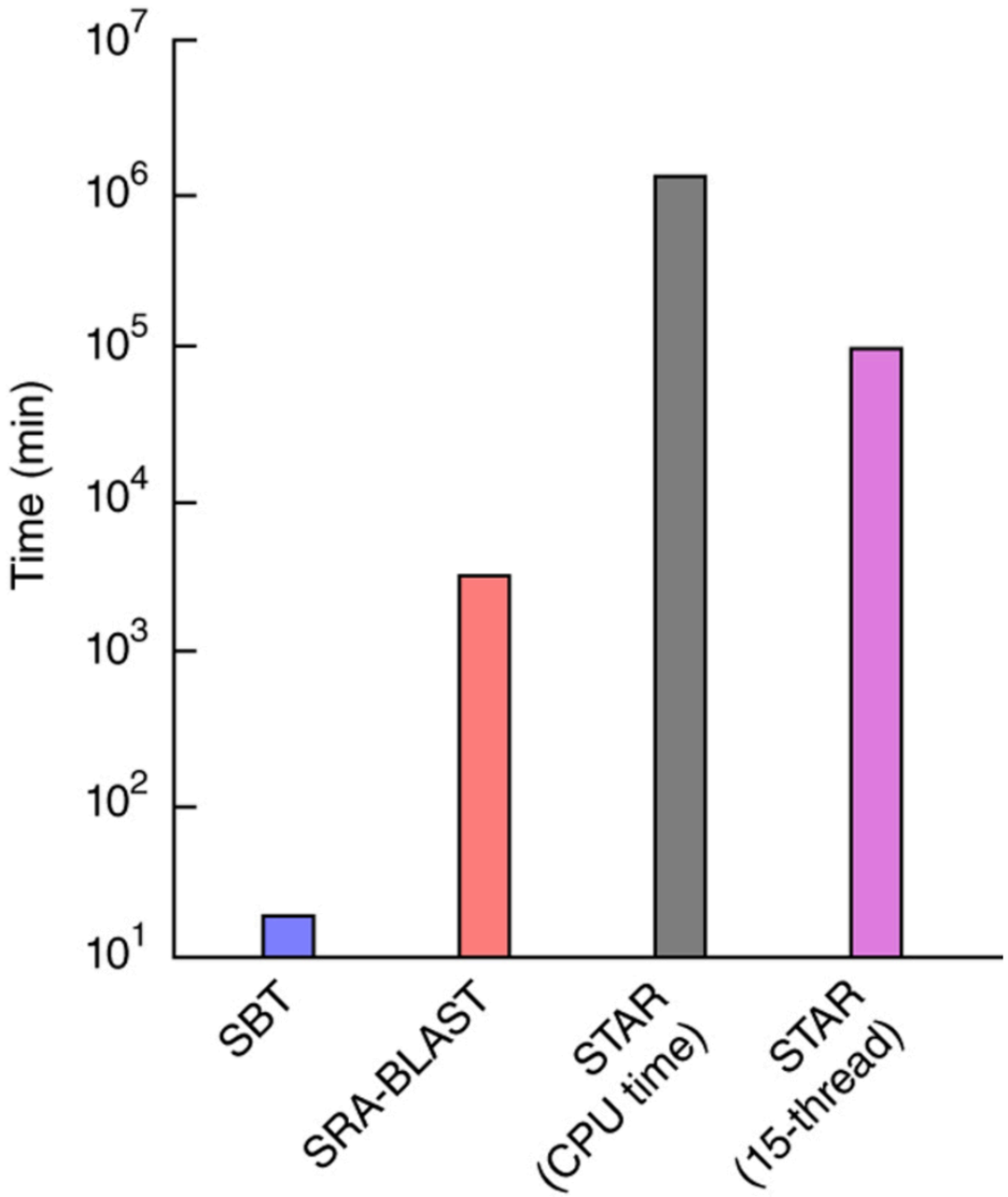
Insert in leaves only k-mers occurring > c times, set

> as follows: $count(s_i) = 1$ if s_i is 300 MB or less, $count(s_i) = 3$ for files of size 300–500 MB, $count(s_i) = 10$ for files of size 500 MB–1 GB, $count(s_i) = 20$ for files between 1 GB and 3 GB, and $count(s_i) = 50$ for files > 3 GB or larger FASTA files.

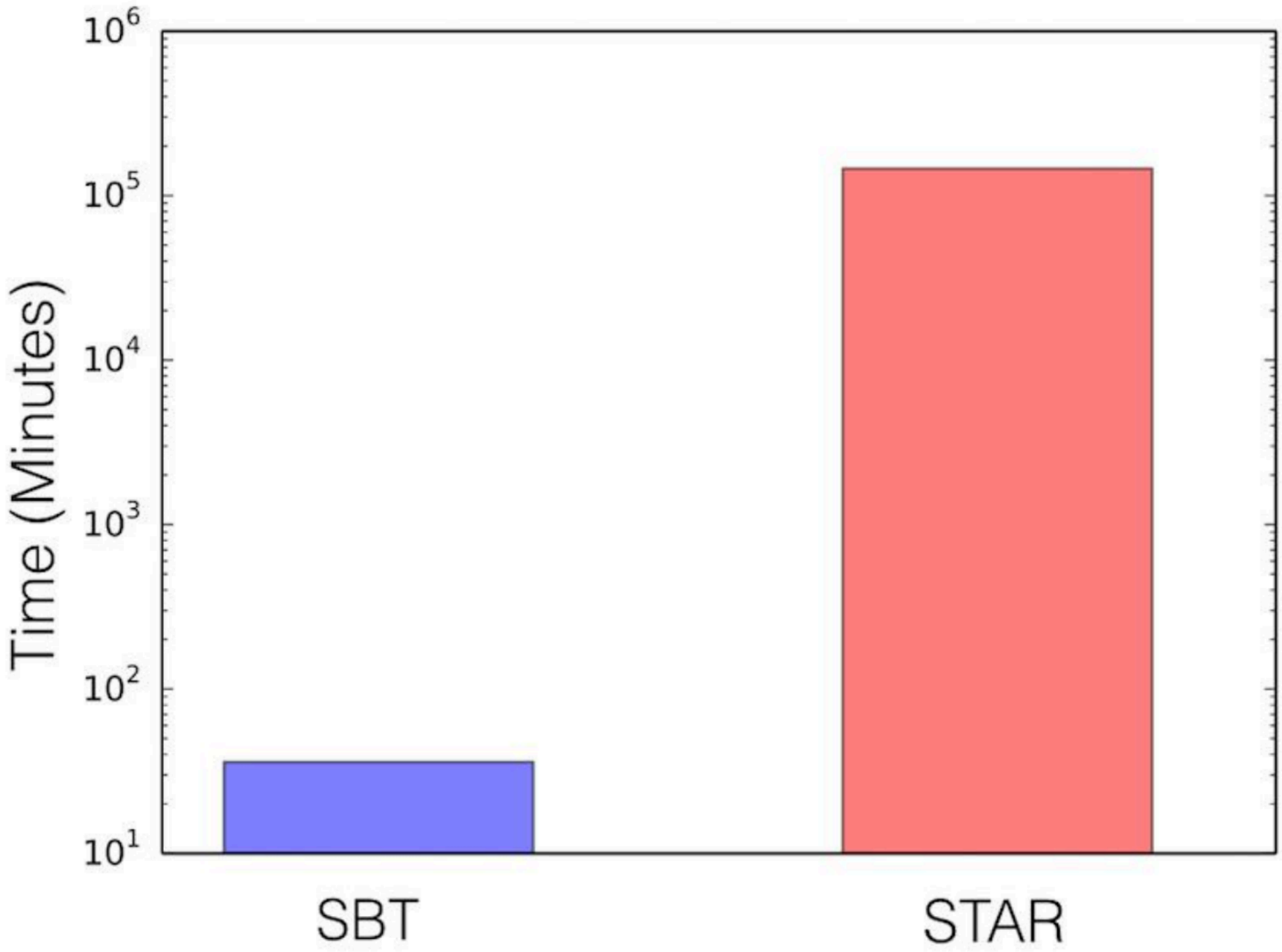
Store Bloom filter as RRR-compressed bit vectors. Greatly reduces storage space. Individual bits can be accessed *without* decompression in O(log m) time.

SBT Speed

Average search time for a single transcript over 2,652 RNA-seq experiments in the SRA for human blood, breast and brain tissues



SBT Speed

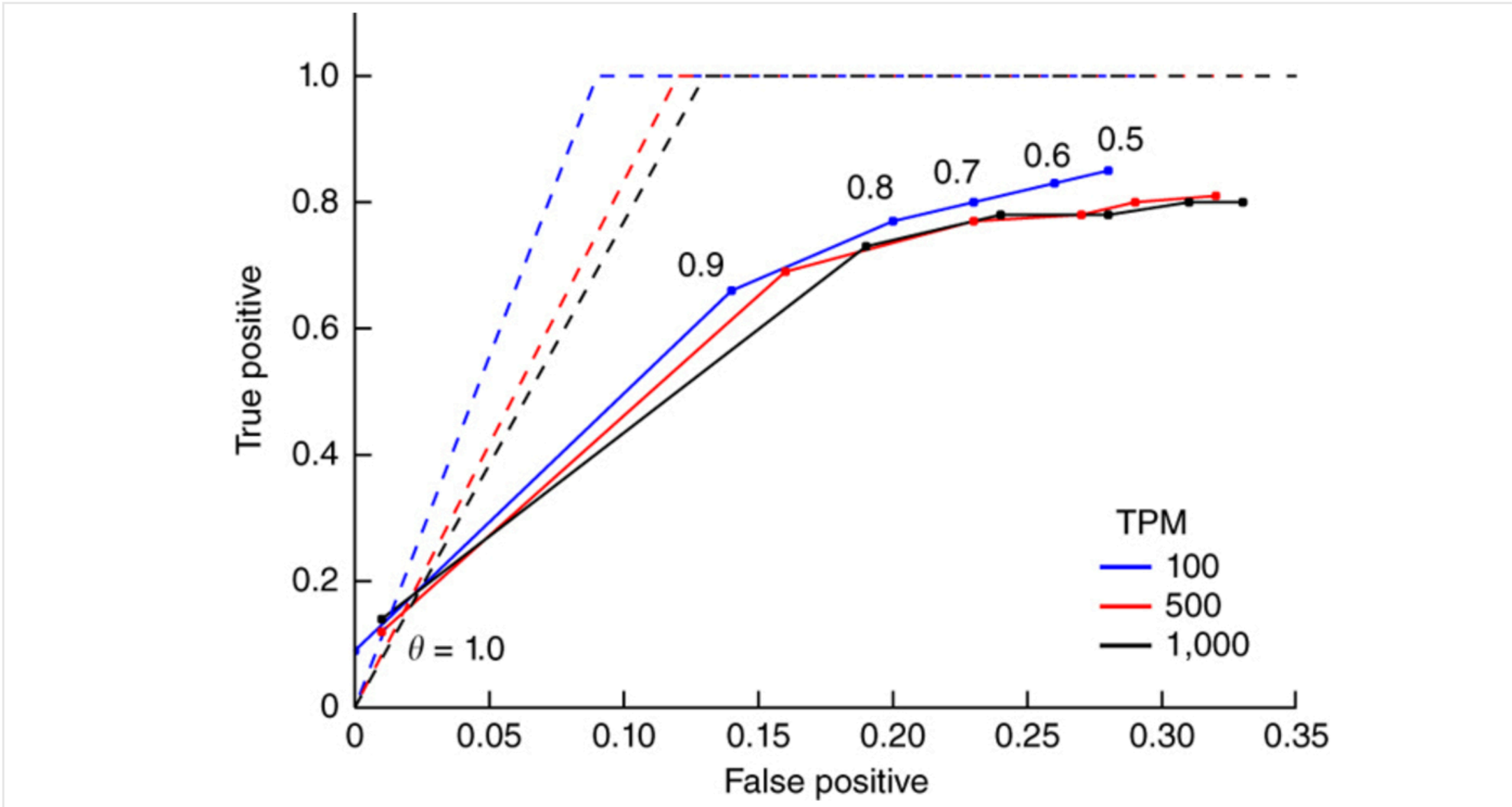


Supplementary Figure 2

Comparison with STAR on batched queries.

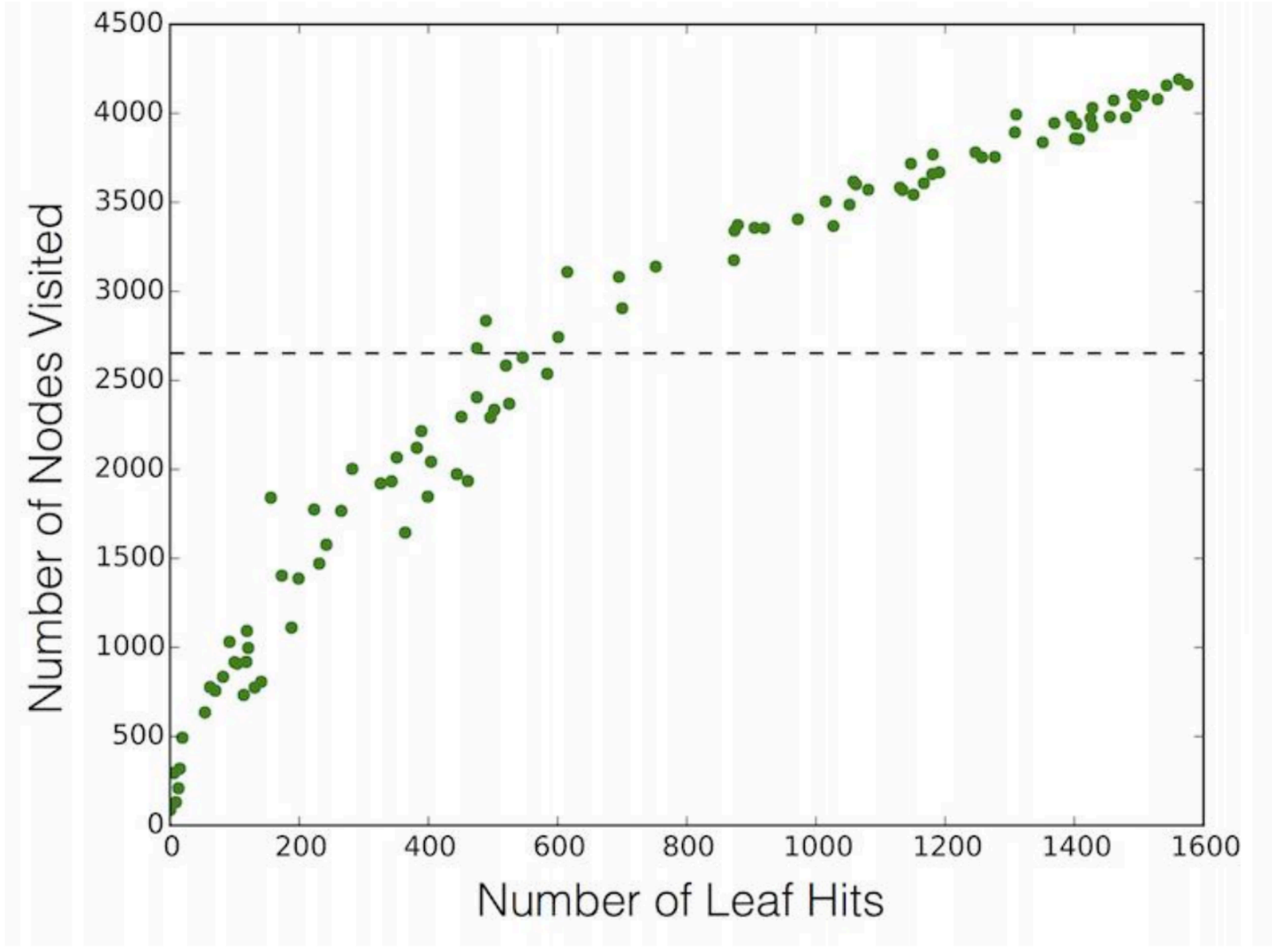
STAR was run using an index built from 100 batch-queries and a size 11 pre-index string. Both SBT and STAR were run using one thread and SBT was limited to a single filter in RAM. SBT is an estimated 4056 times faster than STAR under these conditions. STAR times are estimated from extrapolating from querying 100 random SRR files.

SBT Accuracy



Solid lines represent mean true-positive and false-positive rates, dashed lines represent the median rates on the same experiments. Relaxing θ leads to a higher sensitivity at the cost of specificity. In more than half of all queries, 100% of true-positive hits can be found with θ as high as 0.9.

SBT Efficiency

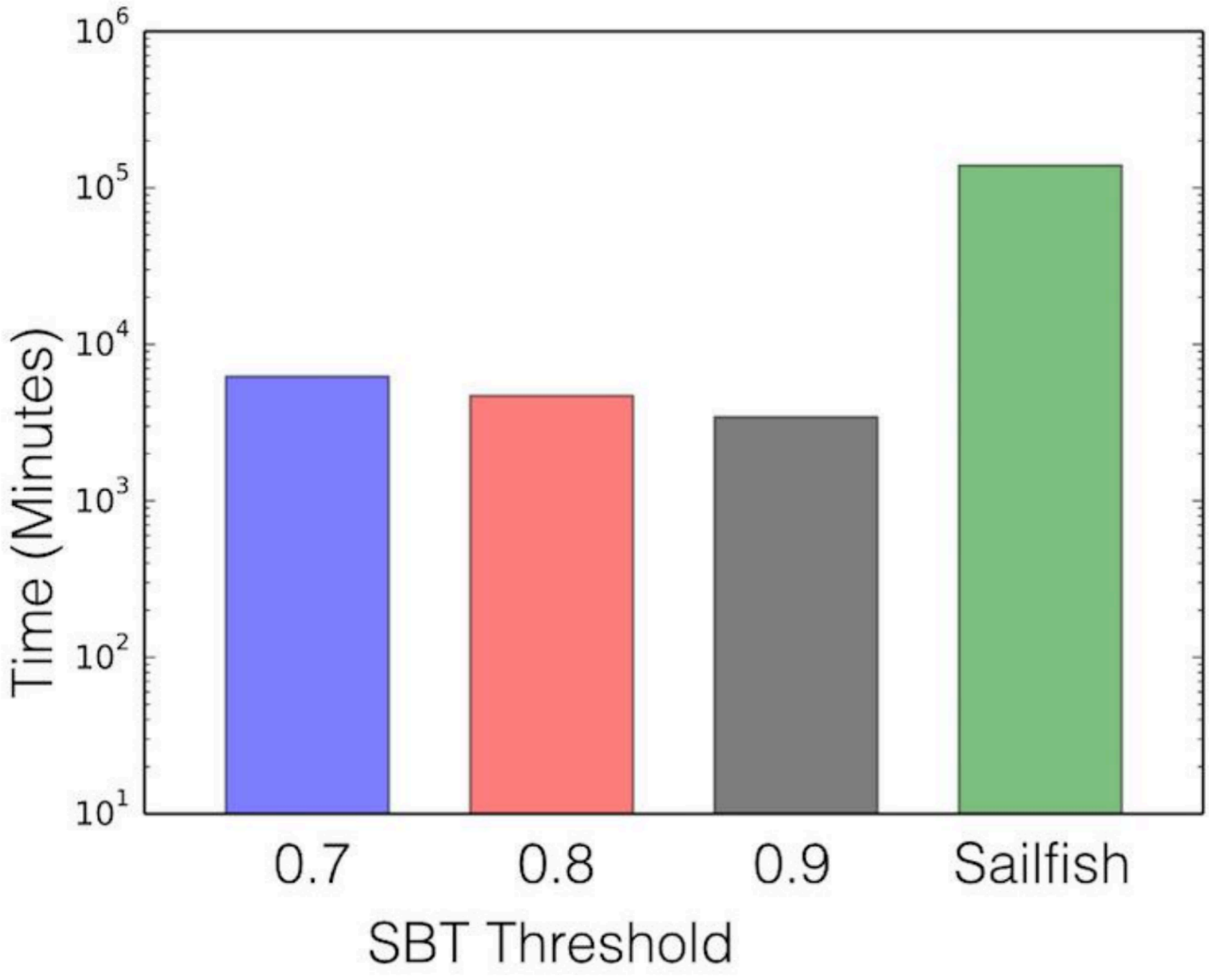


Supplementary Figure 5

Total number of Sequence Bloom Tree nodes visited as a function of the number of leaf hits when querying 100 random human transcripts in the Low query set.

Number of nodes includes both internal and leaf nodes of the SBT. Each point represents a single query. When a query is found in many of the leaves, the query must also visit a nearly equal number of internal tree nodes, and so the tree structure would not provide any benefit over merely searching all the leaf filters directly. On the other hand, when the query is found in only a few leaves, the total number of nodes visited can be significantly smaller than the number of leaves. For the SBT built here, we find that for queries that are found in 600 or fewer leaves, the tree structure and internal nodes result in an improvement of overall efficiency by visiting fewer than 2652 nodes. A naive approach that did not use the tree would require querying 2652 leaf filters for all queries (denoted by dashed line). Approximately half of the randomly selected queries known to be expressed in the included experiments fall below this threshold.

SBT Efficiency



Supplementary Figure 8

Time for querying all known human transcripts.

Total times (single-threaded) for querying all 214,293 human transcripts (in batch mode) against all publicly available blood, breast, and brain RNA-seq experiments in the SRA for θ = 0.7, 0.8, 0.9 as well as the extrapolated time to run Sailfish on the full dataset. Sailfish is significantly faster than nearly all other algorithms for RNA-seq quantification.

Two improved SBT-related papers (RECOMB 2017)

Improved Search of Large Transcriptomic Sequencing Databases Using Split Sequence Bloom Trees

Brad Solomon[1] and Carl Kingsford*[1]

AllSome Sequence Bloom Trees

Chen Sun*[1], Robert S. Harris*[2] Rayan Chikhi[3], and Paul Medvedev†[1,4,5]

Both share a core idea

Split sequence bloom tree (SSBT):

Store **2** filters at each node, r_{sim} and r_{rem}

$r_{sim} = \bigcap_{i=0}^{n} b_i$ present in all leaves below r

$r_{rem} = \bigcup_{i=0}^{n} (b_i - r_{sim})$ present in some (but not) all leaves below r

All Some SBT:

$B_{all}(u) = B_{\cap}(u) \setminus B_{\cap}(parent(u))$ present in all leaves below u, but not in u's parent

$B_{some}(u) = B_{\cup}(u) \setminus B_{\cap}(u)$ present in some (but not) all leaves below u

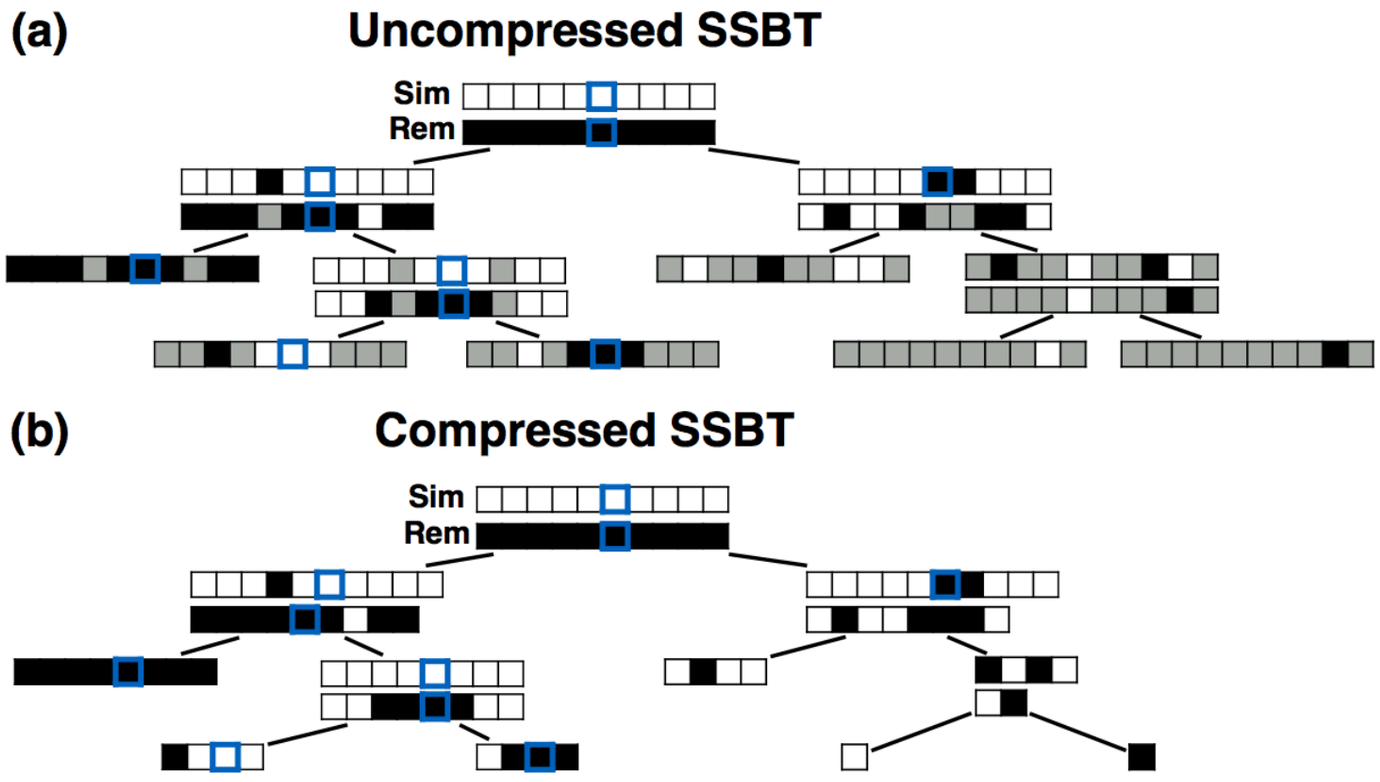
Both share a core idea

This allows an immediate optimization

if r_{sim} or B_{all} match the query, we can add all leaves below r/u without explicitly continuing the search

The details differ

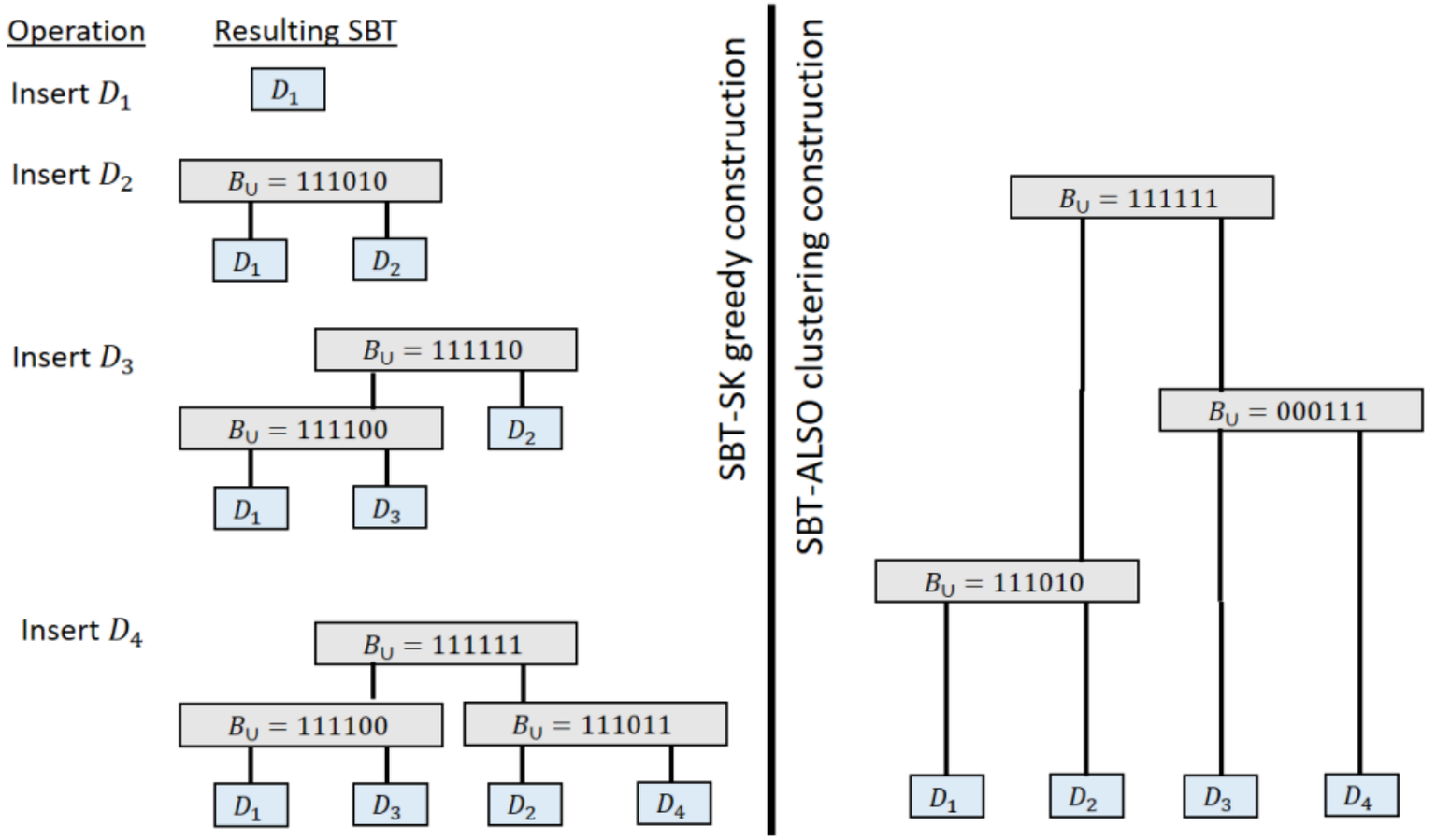
SSBT *explicitly* removed redundant elements



Both share a core idea

The details differ

SBT-AllSome don’t explicitly remove these bits, but they optimize tree construction to put similar filters together (agglomerative clustering)



SSBTs take longer to build than the original but require considerably less memory to store.

Data Index	BFT	SBT	SSBT
Build Time (Min)	195	6	19
Compression Time (Min)	-	6.5	17
Total Time (Min)	**195**	**12.5**	**36**

Table 4: Build and compression times for SBT, SSBT, and BFT constructed from a 50 experiment set. As SBT and SSBT were designed to be queried from a compressed state, we compare the time to build and compress against BFT's time to build.

Data Index	SBT	Split SBT
Build Time	18 Hr	78 Hr
Compression Time	17 Hr	19 Hr
Uncompressed Size	1295 GB	1853 GB
Compressed Size	**200 GB**	**39.7 GB**

Table 2: Build statistics for SBT and SSBT constructed from a 2652 experiment set. The sizes are the total disk space required to store a bloom tree before or after compression. In SSBT's case, this compression includes the removal of non-informative bits.

Data Index	BFT	SBT	SSBT
Build Peak RAM (GB)	23	21.5	15.6
Compress Peak RAM (GB)	-	24.2	16.2
Uncompressed Size (GB)	**9.2**	24	35
Compressed Size (GB)	-	**3.9**	**0.94**

Table 3: Build and compression peak RAM loads and on-disk storage costs for SBT, SSBT, and BFT constructed from a 50 experiment set. BFT does not have a built-in compression tool and cannot be queried when compressed. For these reasons, the uncompressed BFT is compared against the compressed SBT/SSBT.

SSBTs are also faster to query than SBTs

Index	TPM ≥100	TPM ≥500	TPM ≥1000
BFT	75 Sec (11.8 GB)	75 Sec (11.8 GB)	75 Sec (11.8 GB)
SBT	19 Sec (2.9 GB)	21 Sec (3.1 GB)	22 Sec (3.2 GB)
SSBT	5.8 Sec (0.64 GB)	6.2 Sec (0.65 GB)	6.3 Sec (0.66 GB)

Table 5: Comparison in query timing (and average peak memory) between SBT, SSBT, and BFT indices for 50 experiments.

Index	TPM ≥100	TPM ≥500	TPM ≥1000
SBT	19.7 Min	20.7 Min	20 Min
SSBT	3.7 Min	3.8 Min	3.6 Min

Table 6: Comparison in query timing between SBT and SSBT for 2652 experiments.

Query Time:	θ=0.7	θ=0.8	θ=0.9
SBT	20 Min	19 Min	17 Min
SSBT	3.7 Min	3.5 Min	3.2 Min
RAM SSBT	31 Sec	29 Sec	26 Sec

Table 7: Comparison of query times using different thresholds θ for SBT and SSBT using the set of data at TPM 100.

AllSome SBTs are faster to construct than SBT (called SBT-SK here), but not much smaller.

	SBT-SK	SBT-ALSO
construction of tree topology (i.e. clustering)	N/A	27m
construction of internal nodes	56h 54m	26h 3m
temporary disk space	1,235 GB	2,469 GB
final disk space	200 GB	177 GB

Table 1. Construction time and space. Times shown are wall-clock times. A single thread was used. Note the SBT-SK tree that was constructed for the purposes of this Table differs from the tree used in [36] and in our other experiments because the insertion order during construction was not the same as in [36] (because it was not described there).

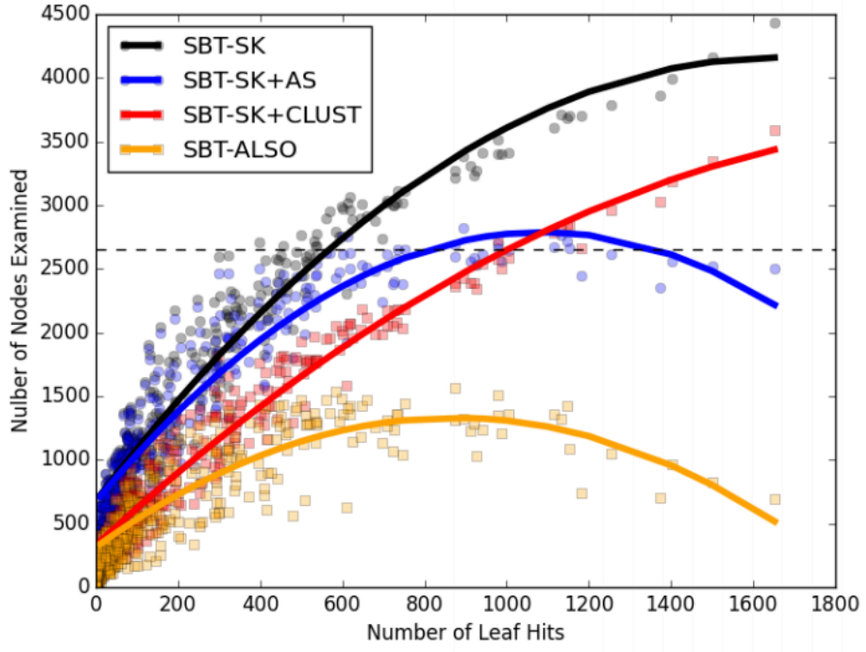


Fig. 3. Number of nodes examined per query for SBT-SK, SBT-ALSO, as well two intermediate SBTs. A set of 1,000 transcripts were chosen at random from Gencode set, and each one queried against the four different trees. A dot represents a query and shows the number of matches in the database (x-axis) compared to the number of nodes that had to be loaded from disk and examined during the search (y-axis). For each tree (color), we interpolated a curve to show the pattern. The dashed horizontal line represents the hypothetical algorithm of simply checking if the query θ-matches against each of the database entries, one-by-one. For θ, we used the default value in the SBT software ($\theta = 0.9$).

They examine fewer nodes than the original SBT too

Which makes them faster to query than the original SBT as well.

	SBT-SK	SBT-SK+CLUST	SBT-ALSO
1 query	1m 11s / 301 MB	56s / 299 MB	34s / 301 MB
10 queries	4m 4s / 305 MB	3m 17s / 304 MB	2m 4s / 313 MB
100 queries	7m 44s / 315 MB	6m 31s / 317 MB	4m 44s / 353 MB
1,000 queries	25m 31s / 420 MB	17m 22s / 418 MB	8m 23s / 639 MB
198,074 queries	3081m 42s / 22 GB	-	462m 39s / 63 GB

Table 2. Query wall-clock run times and maximum memory usage, for batches of different sizes. For the batch of 1,000 queries, we used the same 1,000 queries as in Figure 3. For the batch of 100 queries, we generated three replicate sets, where each set contains 100 randomly sampled transcripts without replacement from the 1,000 queries set. For the batch of 10 queries, we generated 10 replicate sets by partitioning one of the 100 query sets into 10 sets of 10 queries. For the batch of 1 query, we generated 50 replicate sets by sampling 50 random queries from Gencode set. The shown running times are the averages of these replicates. A dash indicates we did not run the experiment. For θ, we used the default value in the SBT software ($\theta = 0.9$).

	SBT-SK	SBT-ALSO		
	regular alg	regular alg	large exact alg	large heuristic alg
query time	1397m 18s	195m 33s	10m 35s	8m 32s
query memory	2.3 GB	4.7 GB	1.3 GB	1.2 GB

Table 3. Performance of different trees and query algorithms on a large query. We show the performance of SBT-SK and three query algorithms using SBT-ALSO compressed with ROAR: the regular algorithm, the large exact algorithm, and the large heuristic algorithm. We show the wall-clock run time and maximum RAM usage. We used $\theta = 0.8$ for this experiment. The ROAR compressed tree was 190 GB (7.3% larger than the RRR tree).

Take home:

SBT “field” is growing quickly

Structure is of both theoretical and practical interest

Some ideas we proposed already done (clustering to build the tree)

Some remain (everyone is still using Bloom filters here)

Mantis: A Fast, Small, and Exact Large-Scale Sequence Search Index

Prashant Pandey, Fatemeh Almodaresi, Michael A. Bender, Michael Ferdman, Rob Johnson, Rob Patro

doi: https://doi.org/10.1101/217372

Posted November 10, 2017.

Mantis uses a fundamentally different idea — no tree

Key idea — Even though many distinct k-mers exist, their appearance in different experiments is not (even close) to independent.

That is; there likely exist a small number of experiment sets that explain most of the k-mers.

More formally, consider an equivalence relation ~ over two k-mers k_1, k_2 such that $k_1 \sim k_2$ iff experiments (k_1) = experiments(k_2)

Total dataset contains ~3.7 billion distinct k-mers.

However, there are only ~222M distinct *color classes*.

Mantis uses a fundamentally different idea — no tree

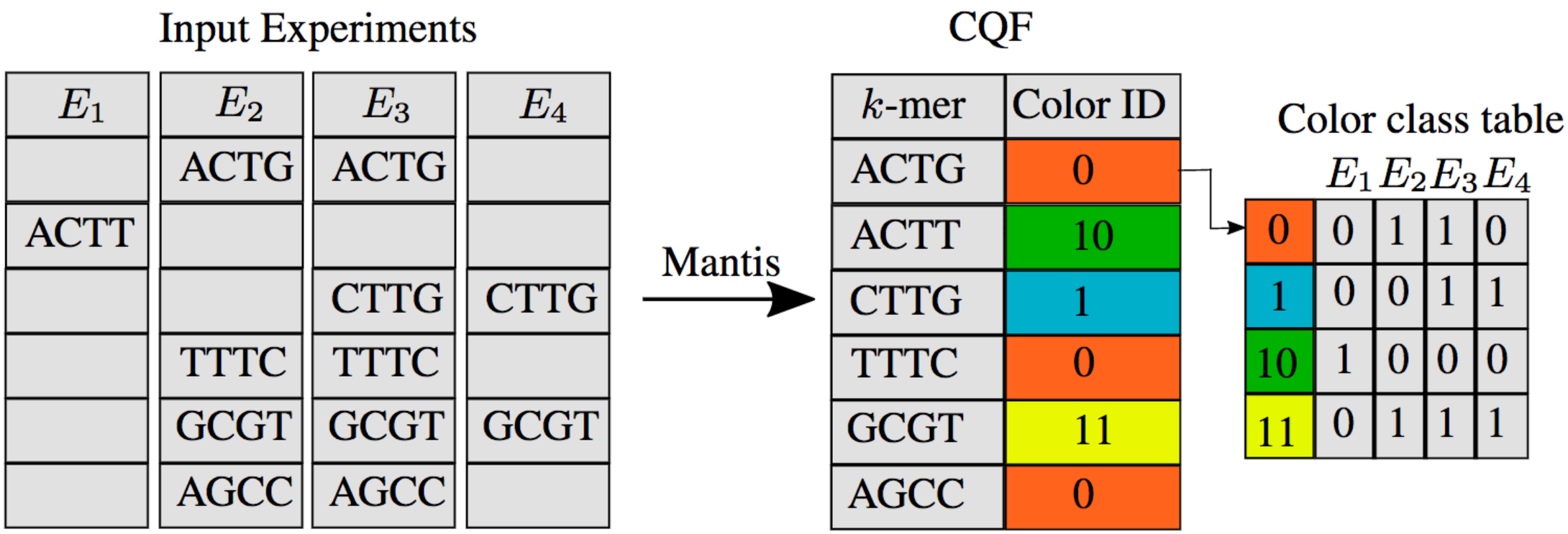


Fig. 1: The Mantis indexing data structures. The CQF contains mappings from k-mers to color-class IDs. The color-class table contains mappings from color-class IDs to bit vectors. Each bit vector is N bits, where N is the number of experiments from which k-mers are extracted. The CQF is constructed by merging N input CQFs each corresponding to an experiment. A query first looks up the k-mer(s) in the CQF and then retrieves the corresponding color-class bit vectors from the color-class table.

Mantis uses a fundamentally different idea — no tree

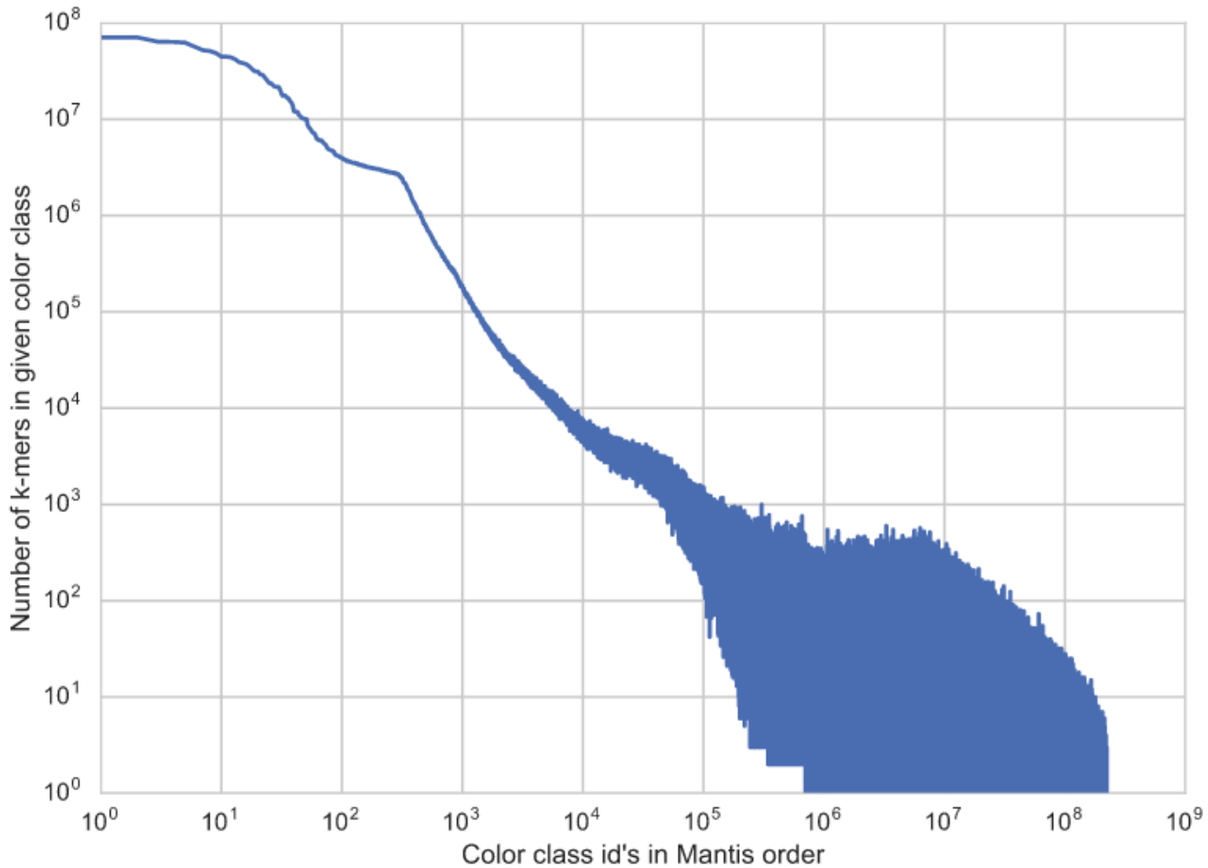


Fig. 2: The distribution of the number of k-mers in a color class and ID assigned to the color class. The color class with the most number of k-mers gets the smallest ID. And the color class with least number of k-mers gets the largest ID. This distribution of IDs helps save space in Mantis.

Mantis is small, and fast …

	Mantis	SSBT
Build time	22 Hr	97 Hr
Representation size	32 GB	39.7 GB

Table 2: Time and space measurement for Mantis and SSBT. Total time taken by Mantis and SSBT to construct the representation. Total space needed to store the representation by Mantis and SSBT. Numbers for SSBT were taken from the SSBT paper [23].

	Mantis	SSBT (0.7)	SSBT (0.8)	SSBT (0.9)
10 Transcripts	25 Sec	3 Min 8 Sec	2 Min 25 Sec	2 Min 7 Sec
100 Transcripts	28 Sec	14 Min 55 Sec	10 Min 56 Sec	7 Min 57 Sec
1000 Transcripts	1 Min 3 Sec	2 Hr 22 Min	1 Hr 54 Min	1 Hr 20 Min

Table 3: Time taken by Mantis and SSBT to perform queries on three sets of transcripts. The set sizes are 10, 100, and 1000 transcripts. For SSBT we used three different threshold values 0.7, 0.8, and 0.9. All the experiments were performed by either making sure that the index structure is cached in RAM or is read from ramfs.

and **exact!**

	Both	Only-Mantis	Only-SSBT	Precision
10 Transcripts	2018	19	1476	0.577
100 Transcripts	22466	146	10588	0.679
1000 Transcripts	160188	1409	95606	0.626

Table 4: Comparison of query benchmark results for Mantis and SSBT. Both means the number of those experiments that are reported by both Mantis and SSBT. Only-Mantis and Only-SSBT means the number of experiments reported by only Mantis and only SSBT. All three query benchmarks are taken from Table 3 for $\theta = 0.8$.

It can be made approximate if we want to save space

Theorem 1. *A query for q k-mers with threshold θ returns only experiments containing at least $\theta q - O(\delta q + \log n)$ queried k-mers w.h.p.*

Proof. This follows from Chernoff bounds and the fact that the number of queried k-mers that are false positives in an experiment is upper bounded by a binomial random variable with mean δq.