Fast and Space-Efficient Maps: Shrinking Big Data Down to Size

Prashant Pandey
Stony Brook University, NY
Sequence Read Archive (SRA) database growth
Currently, sequence searches are restricted to a limited number of experiments. [1]

In this talk

I am going to tell you how to do sequence searches on 10s of terabytes of data.

How to build a space-efficient and performant map and use it to build an index on terabytes of raw sequencing data.
How to attack big data problems
How to attack big data problems

Shrink data to achieve space-efficiency and speed.
How to attack big data problems

Shrink it

- Computational biology
- Streaming applications

Organize it

- File systems

Organize data to achieve efficient disk accesses and better performance.
Research output

Data structure

A General-Purpose Counting Filter: Making Every Bit Count
SIGMOD ‘17

Buffered Count-Min Sketch on SSD: Theory and Experiments
arxiv ‘18

Shrink it

Computational biology

Mantis: A Fast, Small, and Exact Large-Scale Sequence-Search Index
RECOMB ‘18
Cell Systems ‘18

Rainbowfish: A Succinct Colored de Bruijn Graph Representation
WABI ‘17

deBGR: An Efficient and Near-Exact Representation of the Weighted de Bruijn Graph
ISMB ‘17
Bioinformatics ‘17

Organize it

File systems

Squeakr: An Exact and Approximate k-mer Counting System
Bioinformatics ‘17

Writes Wrought Right, and Other Adventures in File System Optimization
TOS ‘16

Optimizing Every Operation in a Write-optimized File System
FAST ‘16

BetrFS: Write-Optimization in a Kernel File System
TOS ‘15

BetrFS: A Right-Optimized Write-Optimized File System
FAST ‘15
Research output

Data structure

Shrink it

Computational biology

Mantis: A Fast, Small, and Exact Large-Scale Sequence-Search Index
RECOMB ’18
Cell Systems ’18

deBGR: An Efficient and Near-Exact Representation of the Weighted de Bruijn Graph
ISMB ’17
Bioinformatics ’17

Rainbowfish: A Succinct Colored de Bruijn Graph Representation
WABI ’17

Squeakr: An Exact and Approximate k-mer Counting System
Bioinformatics ’17

Organize it

File systems

Writes Wrought Right, and Other Adventures in File System Optimization
TOS ’16

Optimizing Every Operation in a Write-Optimized File System
FAST ’16

BetrFS: Write-Optimization in a Kernel File System
TOS ’15

BetrFS: A Right-Optimized Write-Optimized File System
FAST ’15

A General-Purpose Counting Filter: Making Every Bit Count
SIGMOD ’17

Buffered Count-Min Sketch on SSD: Theory and Experiments
arxiv ’18

Computational biology

A General-Purpose Counting Filter: Making Every Bit Count
SIGMOD ’17

Buffered Count-Min Sketch on SSD: Theory and Experiments
arxiv ’18
Research output

Data structure

Shrink it

Computational biology
- Mantis: A Fast, Small, and Exact Large-Scale Sequence-Search Index
  - RECOMB ‘18
  - Cell Systems ‘18
- deBGR: An Efficient and Near-Exact Representation of the Weighted de Bruijn Graph
  - ISMB ‘17
  - Bioinformatics ‘17
- Rainbowfish: A Succinct Colored de Bruijn Graph Representation
  - WABI ‘17

Organize it

File systems
- Writes Wrought Right, and Other Adventures in File System Optimization
  - TOS ‘16
- BetrFS: Write-Optimization in a Kernel File System
  - TOS ‘15
- Optimizing Every Operation in a Write-Optimized File System
  - FAST ‘16
- Squeakr: An Exact and Approximate k-mer Counting System
  - Bioinformatics ‘17
- deBGR: An Efficient and Near-Exact Representation of the Weighted de Bruijn Graph
  - ISMB ‘17
  - Bioinformatics ‘17

SIGMOD ‘17
- A General-Purpose Counting Filter: Making Every Bit Count
- Buffered Count-Min Sketch on SSD: Theory and Experiments
  - arxiv ‘18

arxiv ‘18
- Buffered Count-Min Sketch on SSD: Theory and Experiments

WABI ‘17
- Rainbowfish: A Succinct Colored de Bruijn Graph Representation

Cell Systems ‘18
- Rainbowfish: A Succinct Colored de Bruijn Graph Representation

Bioinformatics ‘17
- Squeakr: An Exact and Approximate k-mer Counting System

TOS ‘15
- BetrFS: A Right-Optimized Write-Optimized File System

FAST ‘15
- BetrFS: A Right-Optimized Write-Optimized File System
Approximate Membership Query (AMQ)

- An AMQ is a lossy representation of a set.
- Operations: inserts and membership queries.
- Compact space:
  - Often taking < 1 byte per item.
  - Comes at the cost of occasional false positives.
The Bloom filter is a bit array + $k$ hash functions.
Bloom filters

The Bloom filter has a bounded false-positive rate.
Bloom filters are ubiquitous

- Streaming applications
- Networking
- Databases
- Computational biology
- Storage systems
Application often must work around Bloom filter limitations.

<table>
<thead>
<tr>
<th>Limitations</th>
<th>Workarounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>No deletions</td>
<td>Rebuild</td>
</tr>
<tr>
<td>No resizes</td>
<td>Guess, rebuild if wrong</td>
</tr>
<tr>
<td>No enumeration</td>
<td>???</td>
</tr>
<tr>
<td>No values</td>
<td>Combine with another data structure</td>
</tr>
</tbody>
</table>
Application often must work around Bloom filter limitations.

<table>
<thead>
<tr>
<th>Limitations</th>
<th>Workarounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>No deletions</td>
<td>Rebuild</td>
</tr>
<tr>
<td>No resizes</td>
<td>Guess, rebuild if wrong</td>
</tr>
<tr>
<td>No enumeration</td>
<td>???</td>
</tr>
<tr>
<td>No values</td>
<td>Combine with another data structure</td>
</tr>
</tbody>
</table>

Bloom filter limitations increase complexity, waste space, and hurt application performance.
The Quotient filter (QF)

- A replacement for the (counting) Bloom filter.
- Space and computationally efficient.
- Can be used as a map for small key-value pairs.
- Uses variable-sized counts/values.

$$QF \text{ space} \leq BF \text{ space} + O(\Sigma |v(x)|)$$
Quotienting: an alternative to Bloom filters

- Store fingerprint compactly in a hash table.
  - Take a fingerprint $h(x)$ for each element $x$.

- Only source of false positives:
  - Two distinct elements $x$ and $y$, where $h(x) = h(y)$.
  - If $x$ is stored and $y$ isn’t, $\text{query}(y)$ gives a false positive.
Storing compact fingerprints

- $b(x) = \text{location in the hash table}$
- $t(x) = \text{tag stored in the hash table}$

Bucket index

Tag

$q$

$r$

$h(x)$

$b(x)$

$t(x)$

$2^q$

0

1

2

3

4

5

6

Storing compact fingerprints

- $b(x) = \text{location in the hash table}$
- $t(x) = \text{tag stored in the hash table}$
Storing compact fingerprints

- $b(x) = \text{location in the hash table}$
- $t(x) = \text{tag stored in the hash table}$

Collisions in the hash table?
Storing compact fingerprints

- $b(x) = \text{location in the hash table}$
- $t(x) = \text{tag stored in the hash table}$

Collisions in the hash table?
- Linear probing.

Bucket index

Tag

$q$

$r$

$h(x)$

$b(x)$

$t(x)$

$2^q$
Storing compact fingerprints

- $b(x) = \text{location in the hash table}$
- $t(x) = \text{tag stored in the hash table}$

Collisions in the hash table?
- Linear probing.
- $O(1)$ metadata bits per slot.

Diagram:
- $h(x)$
- $b(x)$
- $t(x)$
- Bucket index
- Tag

0 1 2 3 4 5 6

$b(x)$

$t(x)$

$t(y)$

$q$

$r$

$2^q$
Resolving collisions in the QF

- CQF uses two metadata bits to resolve collisions and identify the home bucket.

- The metadata bits group tags by their home bucket.
Resolving collisions in the QF

- CQF uses two metadata bits to resolve collisions and identify the home bucket.

- The metadata bits group tags by their home bucket.

\[ t(u) \quad t(v) \quad t(v) \quad t(w) \quad t(x) \quad t(y) \]

insert \( v \)
Resolving collisions in the QF

• CQF uses two metadata bits to resolve collisions and identify the home bucket.

![Diagram showing metadata bits and insert v]

• The metadata bits group tags by their home bucket.

The metadata bits enable us to identify the slots holding the contents of each bucket.
Quotient filter (QF)

Implementation:
2 Meta-bits per slot.

\[ h(x) \rightarrow h_0(x) \parallel h_1(x) \]
Quotient filter (QF)

Implementation:
2 Meta-bits per slot.

\[ h(x) \rightarrow h_0(x) \parallel h_1(x) \]

Abstract Representation

\[ 0 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \]

runends

occupieds

\[ t(a) \]
Quotient filter (QF)

Implementation:
2 Meta-bits per slot.

\[ h(x) \rightarrow h_0(x) \parallel h_1(x) \]

Abstract Representation

\[ 2^q \]

\[ 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \]

\[ h(a) \]

\[ h(b) \]

runends

occupieds

1

1

\[ t(a) \quad t(b) \]
Quotient filter (QF)

Implementation:
2 Meta-bits per slot.

\[ h(x) \rightarrow h_0(x) \| h_1(x) \]

Abstract Representation

\[ \begin{array}{cccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\end{array} \]

\[ h(a) \]
\[ h(b) \]
\[ h(d) \]

runends

occupieds

\[ 1 \]
\[ t(a) \]
\[ t(b) \]
\[ t(d) \]
Quotient filter (QF)

Implementation:
2 Meta-bits per slot.

\[ h(x) \rightarrow h_0(x) \parallel h_1(x) \]

Abstract Representation

```
\begin{align*}
    & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\end{align*}
```

```
\begin{align*}
    & h(a) & h(d) \\
\end{align*}
```

```
\begin{align*}
    & h(b) & h(e) \\
\end{align*}
```

```
\begin{align*}
    & t(a) & t(b) & t(d) & t(e) \\
\end{align*}
```
Quotient filter (QF)

Implementation:
2 Meta-bits per slot.

\[ h(x) \rightarrow h_0(x) \parallel h_1(x) \]

Abstract Representation

0 1 2 3 4 5 6 7

\[
\begin{array}{cccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
h(a) & h(d) \\
h(b) & h(e) \\
h(c) \\
\end{array}
\]

runends

occupieds

\[
\begin{array}{cccccccc}
t(a) & t(b) & t(c) & t(d) & t(e) \\
\end{array}
\]

\[
\begin{array}{cccccccc}
1 & 1 & 1 & 1 & 1 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
2^q & 2^q \\
\end{array}
\]
Quotient filters (QF) use comparable space to Bloom filters (BF).

Bloom filters: $\sim 1.44 \log_2 (1/\varepsilon)$ bits/element.
Quotient filters: $\sim 2.125 + \log_2 (1/\varepsilon)$ bits/element.
Quotient filters (QF) use comparable space to Bloom filters (BF)

Bloom filters: $\sim 1.44 \log_2 \left( \frac{1}{\varepsilon} \right)$ bits/element.
Quotient filters: $\sim 2.125 + \log_2 \left( \frac{1}{\varepsilon} \right)$ bits/element.
Quotient filters (QF) use comparable space to Bloom filters (BF)

\[
\text{Bloom filters: } \sim 1.44 \log_2 (1/\varepsilon) \text{ bits/element.}
\]

\[
\text{Quotient filters: } \sim 2.125 + \log_2 (1/\varepsilon) \text{ bits/element.}
\]

The QF requires less space than the BF for any false-positive rate less than 1/64.
- The QF insert performance in RAM is similar to that of the state-of-the-art non-counting AMQ.
- The QF query performance is significantly fast at low load-factors and slightly slower at higher load-factors.
The QF performance is better or competitive to other non-counting AMQs.

- The QF insert performance in RAM is similar to that of the state-of-the-art non-counting AMQ.
- The QF query performance is significantly fast at low load-factors and slightly slower at higher load-factors.
Quotient filters can also be exact

- Quotient filters store $h(x)$ exactly.
- To store $x$ exactly, use an invertible hash function.

For $n$ elements and $p$-bit hash function:

Space usage: $\sim p \cdot \log_2 n$ bits/element.
Storing key-value pairs

- $b(x) = \text{location in the hash table}$
- $t(x) = \text{tag stored in the hash table}$
- $v(x) = \text{value stored in the hash table}$

- Can store variable-length values.
- $O(1)$ extra metadata bits per slot.
Research output

Data structure

A General-Purpose Counting Filter: Making Every Bit Count
SIGMOD '17

Buffered Count-Min Sketch on SSD: Theory and Experiments
arxiv '18

Shrink it

Mantis: A Fast, Small, and Exact Large-Scale Sequence-Search Index
RECOMB '18
Cell Systems '18

Rainbowfish: A Succinct Colored de Bruijn Graph Representation
WABI '17

Squeakr: An Exact and Approximate k-mer Counting System
Bioinformatics '17

deBGR: An Efficient and Near-Exact Representation of the Weighted de Bruijn Graph
ISMB '17
Bioinformatics '17

Organize it

Computational biology

Writes Wrought Right, and Other Adventures in File System Optimization
TOS '16

Optimizing Every Operation in a Write-optimized File System
FAST '16

File systems

BetrFS: Write-Optimization in a Kernel File System
TOS '15

BetrFS: A Right-Optimized Write-Optimized File System
FAST '15
A huge amount of information is available only in raw sequencing data

- Assembled data is hugely lossy. A lot of variability information is lost during assembly.
- And a lot of raw sequencing data never gets assembled at all.
Current tools (i.e., BLAST) can’t answer diversity questions easily.
Current tools (i.e., BLAST) can’t answer diversity questions easily.

This renders what is otherwise an immensely valuable public resource **largely inert**.
Our Answer: Mantis

- A system to **index and search through large collections of raw sequencing samples**.
- Mantis uses **new data structures** to enable:
  - 6X faster **index construction** than the state-of-the-art.
  - 6X-100X faster **searches** than the state-of-the-art.
  - 20% smaller **index size**.
  - **Exact results**, i.e., no false-positives or -negatives.
- Mantis is also a **colored de Bruijn graph**:
  - Fast graph traversal
  - Topological analyses
The sample discovery problem [Solomon and Kingsford]

Return all samples that contain at least some user-defined fraction $\theta$ of the $k$-mers present in the query string.
The sample discovery problem: solution

- Decompose each sample into $k$-mers.
- If more than $\theta$-fraction $k$-mers from a query appear in a sample then there is a high chance that query appears in that sample.
Existing tools for sample discovery problem

- **SBT**: Solomon and Kingsford 2016
- **SSBT**: Solomon and Kingsford 2017
- **AllSome SBT**: Sun et al. 2017

- All these tools use Bloom filters to represent $k$-mer content of samples.
- Using Bloom filter saves a lot of space but results contain false-positives.
- Also, all these tools have to work around the limitations of Bloom filters.
The SBT is a binary tree of Bloom filter where each leaf node corresponds to a sample.
Sequence Bloom Tree (SBT): Construction

The SBT is a binary tree of Bloom filter where each leaf node corresponds to a sample.
Sequence Bloom Tree (SBT): Construction

The SBT is a binary tree of Bloom filter where each leaf node corresponds to a sample.
Sequence Bloom Tree (SBT): Query

 Traverse the subtree if greater than a $\theta$-fraction $k$-mers are present in the node.
Sequence Bloom Tree (SBT): Query

 Traverse the subtree if greater than a $\Theta$-fraction $k$-mers are present in the node.

$\Theta > 0.75 = \text{NO}$
Sequence Bloom Tree (SBT): Query

Problem: All the Bloom filters must be the same size, but Bloom filters at the top of the tree contain orders of magnitude more items.

Traverse the subtree if greater than a $\theta$-fraction $k$-mers are present in the node.
Mantis: A fundamentally different technique

- We want to map $k$-mers to the samples in which they appear.

<table>
<thead>
<tr>
<th>Input Samples</th>
<th>Map: $k$-mers to Samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>S2</td>
</tr>
<tr>
<td>ACTG</td>
<td>ACTG</td>
</tr>
<tr>
<td>ACTT</td>
<td></td>
</tr>
<tr>
<td>TTC</td>
<td>TTC</td>
</tr>
<tr>
<td>GCGT</td>
<td>GCGT</td>
</tr>
<tr>
<td>AGCC</td>
<td>AGCC</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$k$-mer</th>
<th>Samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTG</td>
<td>S2, S3</td>
</tr>
<tr>
<td>ACTT</td>
<td>S1</td>
</tr>
<tr>
<td>CTTG</td>
<td>S3, S4</td>
</tr>
<tr>
<td>TTC</td>
<td>S2, S3</td>
</tr>
<tr>
<td>GCGT</td>
<td>S2, S3, S4</td>
</tr>
<tr>
<td>AGCC</td>
<td>S2, S3</td>
</tr>
</tbody>
</table>
Mantis: A fundamentally different technique

- There is an inherent redundancy in this design.

<table>
<thead>
<tr>
<th>Input Samples</th>
<th>Map: k-mers to Samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>ACTG</td>
</tr>
<tr>
<td>ACTT</td>
<td>ACTG</td>
</tr>
<tr>
<td></td>
<td>CTTG</td>
</tr>
<tr>
<td></td>
<td>TTTT</td>
</tr>
<tr>
<td></td>
<td>GCGT</td>
</tr>
<tr>
<td></td>
<td>AGCC</td>
</tr>
<tr>
<td>S2</td>
<td></td>
</tr>
<tr>
<td>ACTG</td>
<td>S2, S3</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>ACTT</td>
<td>S1</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>CTTG</td>
<td>S3, S4</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>TTTT</td>
<td>S2, S3</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>GCGT</td>
<td>S2, S3, S4</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>AGCC</td>
<td>S2, S3</td>
</tr>
</tbody>
</table>

Map: k-mers to Samples

<table>
<thead>
<tr>
<th>k-mer</th>
<th>Samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTG</td>
<td>S2, S3</td>
</tr>
<tr>
<td>ACTT</td>
<td>S1</td>
</tr>
<tr>
<td>CTTG</td>
<td>S3, S4</td>
</tr>
<tr>
<td>TTTT</td>
<td>S2, S3</td>
</tr>
<tr>
<td>GCGT</td>
<td>S2, S3, S4</td>
</tr>
<tr>
<td>AGCC</td>
<td>S2, S3</td>
</tr>
</tbody>
</table>
Mantis: A fundamentally different technique

- We add another layer of indirection from IDs to sets of samples.

<table>
<thead>
<tr>
<th>Input Samples</th>
<th>Map: $k$-mers to IDs</th>
<th>Map: IDs to Samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACTG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACTG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACTG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACTG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACTG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACTG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACTG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACTG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACTG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACTG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACTG</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- $k$-mers to IDs:
  - ACTG: ID 0
  - ACTT: ID 10
  - CTTG: ID 1
  - TTTC: ID 0
  - GCGT: ID 11
  - AGCC: ID 0

- IDs to Samples:
  - 0: S2, S3
  - 1: S3, S4
  - 10: S1
  - 11: S2, S3, S4
Mantis: A fundamentally different technique

In Mantis, we build a map from $k$-mer to the set of samples in which they appear.
Mantis: A fundamentally different technique

In Mantis, we use the exact version of the QF to map \( k \)-mers to variable-sized color class IDs.
Mantis: A fundamentally different technique

In Mantis, we use the exact version of the QF to map k-mers to variable-sized color class IDs.
Experimental setup

- Build index for 2652 samples of RNA-seq short-read sequencing runs of human blood, brain, and breast tissue.
- Total size of all the samples: ≈7 Terabytes
- Compared with SSBT.
- Evaluation metrics:
  - Index size
  - Construction time
  - Query performance
  - Quality of results
Mantis is 20% smaller than the SSBT.
Mantis is 6X faster than the SSBT for construction.
Mantis is 6X-100X faster than the SSBT for queries.
Quality of results

θ = 0.9 for SSBT

Mantis is exact.
Quality of results

\[ \theta = 0.9 \] for SSBT

A space efficient map yields a faster, smaller, simpler, and an exact solution to the sample discovery problem

Mantis is exact.
Proposed work: Improving the QF insertion throughput at high load factors
Insertion throughput at high load factors

Insertion throughput at 90% load factor is \(5\times - 10\times\) slower than at 10% load factor.

![Inserts](image)
Improving insertion throughput at high load factors

- Use a small stash to keep a check on cluster lengths.
  - Use a stash of size $\approx (n/\log n)$ slots.
  - If the distance between the home slot and the next empty slot is greater than 64 (or 128), use the stash.

- Cache align qf-blocks.

- Give up counting to reduce shifting of tags.
  - No need to keep tags in orders inside runs.
Target performance profile (TPP)

![Graph showing insertions over load factor with different profiles: CQF, CF, and TPP.](image)
Target performance profile (TPP)

We want a more graceful degradation
Conclusion

- Space-efficient and fast compact data structures can help solve big data problems across subfields.

- We need to redesign applications to get complete benefits of today’s feature-rich AMQs.

Source code: https://github.com/splatlab/