Compact Representations of Annotated de Bruijn Graphs

Prashant Pandey
Stony Brook University, NY, USA
De Bruijn graphs are ubiquitous

A de Bruijn graph is the data representation at the heart of a lot of sequence analyses.

[De Bruijn graphs are ubiquitous]

Raw sequencing data

De Bruijn graph

Sequence search

Short/Long reads transcriptome assembly

Long reads error correction
De Bruijn graph (dBG)

An edge is a length-k string connecting its two (k-1) substrings.
De Bruijn graph (dBG)

A **read** is a string of bases over the DNA alphabet A, C, T, and G.

A **k-mer** is a substring of length k. Here, k is 5.
De Bruijn graph (dBG)

Read 1: ....CAAAAAT....
Read 2: ....CAAAAAC....
Annotated De Bruijn graphs

- Topology-only de Bruijn graphs are not adequate for downstream applications.

- Abundance information of each k-mer is critical for transcriptome assembly.

- Information about samples in which a k-mer is present in a union de Bruijn graph of multiple samples is critical for variant discovery.
A weighted de Bruijn graph associates each edge (k-mer) its abundance in the underlying dataset.
Colored de Bruijn graph (CdBG)

Sample 1: ….CAAAAAT….  
Sample 2: ….CAAAAAC…..

A colored de Bruijn graph is a union graph of multiple samples, where the identity of each sample is retained by coloring those edges present in a sample.
Measuring annotated dBG representation

deo Bruijn graphs store only k-mers, memory usage scales with the number of unique k-mers.

Human genome (few Billion k-mers): >100 GB
Soil metagenomes (few Million species): Few TBs

Beefy server machines are needed to perform weighted de Bruijn graph analysis.
In this talk

• A compact representation of annotated de Bruijn graphs.
  • It would enable transcriptome assembly on machines with less resources.
  • It would also enable assembly of fundamentally large datasets that wasn’t possible before on a single machine.
  • It would also enable sequence-level searches and variation detection on all of the available RNA-seq experiments in SRA.
dBG as a set

<table>
<thead>
<tr>
<th>Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCCG</td>
</tr>
<tr>
<td>CCGC</td>
</tr>
<tr>
<td>CCGA</td>
</tr>
<tr>
<td>CGCT</td>
</tr>
<tr>
<td>AGCT</td>
</tr>
</tbody>
</table>

(Edges)

de Bruijn graph
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.
Probabilistic de Bruijn graph

[Bell et al. 2012]

Representing a dBG using a Bloom filter.

Topological errors
Probabilistic de Bruijn graph

[Bellow et al. 2016]

Bloom filter

TCCG
CCGC
CCGA
CGCT
AGCT
GAGC
CGAG
GAGT
AAAT

Topological errors
Showed how to exploit redundancy in k-mers to reduce the false-positive rate of the Bloom filter without increasing the space.
Exact de Bruijn graph

[Chikhi and Rizk 2013] and [Salikhov et al. 2013]

Critical false-positive k-mers

They showed how to convert a probabilistic representation into an exact one using a small and exact auxiliary data structure.
WdBG as a multiset

<table>
<thead>
<tr>
<th>MultiSet</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>TCCG, 2</td>
<td></td>
</tr>
<tr>
<td>CCGC, 9</td>
<td></td>
</tr>
<tr>
<td>CCGA, 6</td>
<td></td>
</tr>
<tr>
<td>CGCT, 5</td>
<td></td>
</tr>
<tr>
<td>AGCT, 2</td>
<td></td>
</tr>
</tbody>
</table>

(Edge, Abundance)

Weighted de Bruijn graph
Counting filters: AMQs for multisets

- A counting filter is a lossy representation of a multiset.
- Operations: inserts, count, and delete.
- Generalizes AMQs
  - False positives $\approx$ over-counts.
- Counting quotient filter
The counting quotient filter

[Pandey et al. SIGMOD 2017]

- Smaller than many non-counting AMQs
  - Bloom, cuckoo [Fan et al., 2014], and quotient [Bender et al., 2012] filters.
- Uses variable-sized counters to handle skewed data sets efficiently.
- Good cache locality
- Deletions
- Dynamically resizable
- Mergeable
Squeakr

[Pandey et al. BIOINFORMATICS 2017]

Topological errors

Abundance error

Counting quotient filter

TCCG, 4
CCGC, 9
CCGA, 6
CGCT, 5
AGCT, 4
GAGC, 2
CGAG, 1
GAGT, 1
AAAT, 1

Approximate weighted de Bruijn graph.
deBGR
[Pandey et al. ISMB 2017]

• An exact representation of the weighted de Bruijn graph.

• An algorithm that uses counts in the approximate representation in an AMQ to iteratively self-correct approximation errors.

• It corrects both kinds of errors, abundance and topological errors and supports membership queries.

• It takes 18-28% more space than the approximate representation and has no errors.
A weighted de Bruijn graph invariant

Read 1: ….CAAAAAT….  
Read 2: ….CAAAAAC….  

Total incoming abundance = Total outgoing abundance
A weighted de Bruijn graph invariant

Total incoming abundance = Total outgoing abundance*

*After accounting for read starts and ends.
WdBG representation in deBGR

Read 1: CAAAAT

Read 2: CAAAAC

<table>
<thead>
<tr>
<th>Edge</th>
<th>Abundance</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAAA</td>
<td>2</td>
</tr>
<tr>
<td>AAAAT</td>
<td>1</td>
</tr>
<tr>
<td>AAAAC</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Node</th>
<th>Start reads</th>
<th>End reads</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAAA</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>AAAAT</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>AAAC</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>
WdBG representation in deBGR

<table>
<thead>
<tr>
<th>Edge</th>
<th>Abundance</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAAA</td>
<td>2</td>
</tr>
<tr>
<td>AAAAT</td>
<td>2</td>
</tr>
<tr>
<td>AAAAC</td>
<td>1</td>
</tr>
<tr>
<td>CCGTA</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Node</th>
<th>Start reads</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAAA</td>
<td>2</td>
</tr>
<tr>
<td>AAAAT</td>
<td>1</td>
</tr>
<tr>
<td>AAAC</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Node</th>
<th>End reads</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAAT</td>
<td>0</td>
</tr>
<tr>
<td>AAAC</td>
<td>0</td>
</tr>
</tbody>
</table>

- CCGT: Start reads: 0, End reads: 0
- CCGTA: Start reads: 0, End reads: 1
- CGTA: Start reads: 0, End reads: 0
- AAAT: Start reads: 0, End reads: 1
- CAAA: Start reads: 2, End reads: 0
- AAAAC: Start reads: 0, End reads: 1
- AAAA: Start reads: 0, End reads: 0
- AAAC: Start reads: 0, End reads: 1

Graph edges and node connections:
- CCGT → CCGTA
- CCGTA → CGTA
- CCGTA → AAAAT
- AAAAT → AAAC
- AAAA → AAAC
- CAAA → AAAA
- CAAA → AAAAT
- AAAAT → AAAAC
Error correction

- **CCGT**
  - Start reads: 0
  - End reads: 0

- **CGTA**
  - Start reads: 0
  - End reads: 0

- **AAAA**
  - Start reads: 0
  - End reads: 0

- **AAAT**
  - Start reads: 0
  - End reads: 1

- **AAAC**
  - Start reads: 0
  - End reads: 1

- **CCGT**
  - Start reads: 0
  - End reads: 0

- **CAAA**
  - Start reads: 2
  - End reads: 0

- **AAAA**
  - Start reads: 0
  - End reads: 0

- **AAAT**
  - Start reads: 0
  - End reads: 1

- **AAAAC**
  - Start reads: 0
  - End reads: 1

- **AAAC**
  - Start reads: 0
  - End reads: 1

- **CAAAA**
  - Start reads: 2
  - End reads: 0

- **AAAAT**
  - Start reads: 0
  - End reads: 1

- **AAAAC**
  - Start reads: 0
  - End reads: 1
Error correction

- **CCGT** (Start reads: 0, End reads: 0)
  - CCGTA, + 0

- **CGTA** (Start reads: 0, End reads: 0)
  - CCGTA

- **AAAT** (Start reads: 0, End reads: 1)
  - AAAAT, 0
  - AAAAT

- **AAAA** (Start reads: 0, End reads: 0)
  - CAAAAA, 2
  - CAAAA

- **AAAAA** (Start reads: 0, End reads: 0)
  - AAAA

- **AAAC** (Start reads: 0, End reads: 1)
  - AAAAC, 1

- **AAAC** (Start reads: 0, End reads: 1)
  - AAAAC, 2

- **AAAC** (Start reads: 0, End reads: 1)
  - AAAAC, 1

Error correction processes involve transitions between states, as indicated by the arrows.
Error correction
Error correction

- **CCGT**
  - Start reads: 0
  - End reads: 0

- **CGTA**
  - Start reads: 0
  - End reads: 0

- **AAAT**
  - Start reads: 0
  - End reads: 1

- **AAAC**
  - Start reads: 0
  - End reads: 1

- **CAAA**
  - Start reads: 2
  - End reads: 0

- **AAAA**
  - Start reads: 0
  - End reads: 0

- **CAAAT**
  - Start reads: 0
  - End reads: 1

- **AAAC**
  - Start reads: 0
  - End reads: 1
Error correction

CCGT
Start reads: 0
End reads: 0

CGTA
Start reads: 0
End reads: 0

AAAA
Start reads: 0
End reads: 0

AAAT
Start reads: 0
End reads: 1

AAAC
Start reads: 0
End reads: 1
Error correction

- **CCGT**
  - Start reads: 0
  - End reads: 0

- **CGTA**
  - Start reads: 0
  - End reads: 0

- **AAAT**
  - Start reads: 0
  - End reads: 1

- **AAAA**
  - Start reads: 0
  - End reads: 0

- **AAAC**
  - Start reads: 0
  - End reads: 1

- **CAAA**
  - Start reads: 2
  - End reads: 0

- **AAAAA**
  - Start reads: 2
  - End reads: 0

- **CAAAA**
  - Start reads: 0
  - End reads: 0

- **AAAAT**
  - Start reads: 0
  - End reads: 1
Error correction

- **CCGT**
  - Start reads: 0
  - End reads: 0
  - CCGTA, + 0

- **CGTA**
  - Start reads: 0
  - End reads: 0

- **AAAT**
  - Start reads: 0
  - End reads: 1
  - AAAT, 2

- **AAAA**
  - Start reads: 0
  - End reads: 0
  - AAAA, 2

- **AAAC**
  - Start reads: 0
  - End reads: 1
  - AAAC, 1

- **CAAA**
  - Start reads: 2
  - End reads: 0
  - CAAAA, 2

- **AAAAC**
  - Start reads: 0
  - End reads: 1

- **AAAC**
  - Start reads: 0
  - End reads: 1
Error correction

- **CCGT**
  - Start reads: 0
  - End reads: 0

- **CGTA**
  - Start reads: 0
  - End reads: 0

- **AAAT**
  - Start reads: 0
  - End reads: 1

- **AAAC**
  - Start reads: 0
  - End reads: 1

- **AAAA**
  - Start reads: 0
  - End reads: 0

**Error correction**

- **CCGTA**
  - Start reads: 0
  - End reads: 0

- **AAAAA**
  - Start reads: 2
  - End reads: 0

- **AAAA**
  - Start reads: 0
  - End reads: 0

- **AAAAC**
  - Start reads: 0
  - End reads: 1
Error correction

- **CCGT**
  - Start reads: 0
  - End reads: 0
  - CCGTA, + 0

- **CGTA**
  - Start reads: 0
  - End reads: 0

- **AAAAT**
  - Start reads: 0
  - End reads: 1
  - AAAAT, 2, 1

- **AAAAC**
  - Start reads: 0
  - End reads: 1
  - AAAAC, 1

- **AAAA**
  - Start reads: 0
  - End reads: 0
  - CAAAAA, 2

- **AAAC**
  - Start reads: 0
  - End reads: 1
  - AAAC, 1

- **CAAA**
  - Start reads: 2
  - End reads: 0

- **AAAAA**
  - Start reads: 0
  - End reads: 0
  - CAAAAA, 2
Error correction algorithm

• We use a standard work queue algorithm.

• We bootstrap with a set $C$ of edges for which we know the abundance is correct.

• We then expand the set $C$ of edges using the weighted de Bruijn graph invariant.

• Please refer to the paper for exact set of rules for error correction.

• Running time: $O(n \cdot \log(n)/\log(1/4\varepsilon))$. 
## Datasets

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Size</th>
<th>#k-mer instances</th>
<th>#Distinct k-mers</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSM984609</td>
<td>26 GB</td>
<td>19,662,773,330</td>
<td>1,146,347,598</td>
</tr>
<tr>
<td>GSM981256</td>
<td>22 GB</td>
<td>16,470,774,825</td>
<td>1,118,090,824</td>
</tr>
<tr>
<td>GSM981244</td>
<td>43 GB</td>
<td>37,897,872,977</td>
<td>1,404,643,983</td>
</tr>
<tr>
<td>SRR1284895</td>
<td>33 GB</td>
<td>26,235,129,875</td>
<td>2,079,889,717</td>
</tr>
</tbody>
</table>
Space vs Accuracy

- **Datasets**
  - GSM984609
  - GSM981256
  - GSM981244
  - SRR1284895

- **Graph Details**
  - X-axis: Datasets
  - Y-axis: Bits/k-mer
  - Colors: Squeakr (blue), Squeakr (exact) (green), deBGR (yellow)
Space vs Accuracy

Datasets

- GSM984609: 16,655,318
- GSM981256: 15,864,754
- GSM981244: 12,257,261
- SRR1284895: 27,200,821
Space vs Accuracy

Number of errors in deBGR: 0!

Datasets:
- GSM984609
- GSM981256
- GSM981244
- SRR1284895

Bits/k-mer:
- GSM984609: 16,655,318
- GSM981256: 15,864,754
- GSM981244: 12,257,261
- SRR1284895: 27,200,821
Conclusion

• Abundance information is important for many data analyses.

• But the abundance information can be used to remove effectively all the errors in an approximate weighted de Bruijn graph representation.

• The basic ideas behind our error-correction technique may also be useful for compactly representing weighted graphs by exploiting other domain-specific invariants.

https://github.com/splatlab/
De Bruijn graph (dBG)

In graph theory, an \textit{n-dimensional} de Bruijn graph of \textit{m} symbols is a directed graph representing overlaps between sequences of symbols.
The counting quotient filter (CQF)  
[Pandey et al. SIGMOD 2017]

- A replacement for the (counting) Bloom filter.
- Space and computationally efficient.
- Uses variable-sized counters to handle skewed data sets efficiently.

$$\text{CQF space} \leq \text{BF space} + O\left(\sum_{x \in S} \log c(x)\right)$$

Asymptotically optimal
Counting quotient filter (CQF)

- Smaller than many non-counting AMQs
  - Bloom, cuckoo [Fan et al., 2014], and quotient [Bender et al., 2012] filters.
- Good cache locality
- Deletions
- Dynamically resizable
- Mergeable
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

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

- \( b(x) = \) location in the hash table
- \( t(x) = \) tag stored in the hash table

Collisions in the hash table?

Linear probing.
Storing compact fingerprints

Does \( t(v) \) belong to bucket 4 or 5?

- The home bucket for \( t(u) \) and \( t(v) \) is 4.
Resolving collisions in the CQF

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

- The metadata bits group tags by their home bucket.
Encoding counts

- Metadata scheme tells us the run of slots holding contents of a bucket.
- We can encode contents of buckets however we want.
- *The original quotient filter used repetition (unary).*
We want to count in binary, not unary.

Idea: use some of the space for tags to store counts.

Issue: determine which are tags and which are counts without using even one “control” bit.

Encoding counts

```
1 1
\text{t(u)} 4 \text{t(x)} 1 \text{t(y)} 1
```

4 copies of $t(u)$
**Performance: In memory**

- The CQF insert performance in RAM is similar to that of state-of-the-art **non-counting** AMQs.
- The CQF is significantly faster at low load factors and slightly slower on high load factors.
Performance: Skewed datasets

- The CQF outperforms the CBF by a factor of 6x-10x on both inserts and lookups.