Compacting de Bruijn graphs from sequencing data quickly and in low memory

Rayan Chikhi (CNRS)

joint work with A. Limasset (ENS Rennes), P. Medvedev (Penn State)

ISMB 2016
de Bruijn Graph

sequence: GATTACATTACAA
k-mers: GAT
(k=3) ATT
TTA

nodes: k-mers (words of length k)
edges: exact suffix-prefix overlaps of length k – 1

- assembly of genomes, metagenomes
- variant calling
- RNA-seq assembly & quantification
Compacted de Bruijn Graph

non-compacted de Bruijn graph:

```
TCA → CAT → ATT → TTG
          ↑   ↑   ↑
TGG → GGT → GTA → TAA
          ↑   ↑   ↑
TGC → GCG → CGA → GAA
          ↑   ↑   ↑
AAC → ACC → CCG
```

Compacted de Bruijn graph:

```
TCATTG
     ↓
TGGTAA
     ↓
TGCGAA
     ↓
AACCG
```

Each non-branching path becomes a single node (unitig).

- no loss of information
- less space
Steps of de Bruijn graph assemblers

1.1 TB reads.gz

- k-mer counting

- graph compaction

- graph cleaning

Input data
20 Gbp spruce [Birol 2013]

1.1 TB

700 GB k-mers

30 GB unitigs

Recent progress,
Stand-alone software
(KMC2, DSK2, Jellyfish2)

Integrated in assemblers,
high-memory or slow
This work

- computationally intensive
- bottlenecks at early stages

Integrated in assemblers,
Heuristics
20 Gbp spruce and 22 Gbp pine

Previous assemblies
- spruce: 2 days, 1380 cores, 4.3 TB RAM [Birol 2013]
- pine: 3 months, 32 cores, 0.8 TB RAM [Zimin 2014]

This work:
improve performance by **orders of magnitude** (up to compaction step)
BCALM 2

Software for constructing and compacting de Bruijn graphs

Successor of BCALM 1 (single-threaded)

Parallel graph compaction is non-trivial, let’s see why..
Parallel compaction, first attempt

**Input k-mers**
partitioned on disk, based on minimizers

1-thread classical compaction

**minimizer** of $s$: smallest $\ell$-mer in $s$

[Roberts et al, 2004]

e.g. ($\ell = 2$, lexicographical order)

- TGACGGG
- GACGGGT
- ACGGGTC
- CGGGTCA
- GGGTCAG
- GGTCAGA

Frequency ordering → better repartition.
[RECOMB’14]
Compaction of partitions

Unitigs:

GTGATGA
ATGACC
ATGAACT

$k$-mers are partitioned w.r.t minimizer.
In this case, compacting all partitions returns exactly all the unitigs.
This case indicates that partitions contain sub-strings of unitigs. Those substrings need to be later merged.
2-step strategy

Input k-mers

Parallel partial compaction algorithm

Intermediate file

Parallel glue algorithm

Unitigs
Simple partitioning is not enough

Compacting partitions may create false unitigs (due to missing edges).

- A simple fix: put certain $k$-mers into two partitions.
- $x$ is a **doubled kmer** when
  $\text{minimizer}(x[1..k-1]) \neq \text{minimizer}(x[2\ldots k])$. 

$$
\begin{align*}
\text{GTGAC} & \quad \text{TGACG} & \quad \text{AC} \\
\text{AC} & \quad \text{GACGA} & \quad \text{GACGA} \\
\text{AC} & \quad \text{ACGAC} & \quad \text{ACGAA} \\
\text{AC} & \quad \text{CGAAG} & \quad \text{AA}
\end{align*}
$$
**BCALM 2’s partial compaction module**

**Doubled kmers**
are inserted in
two partitions

1-thread classical compaction

---

**Lemma 1:**
doubled $k$-mers appear as prefixes or suffixes of compacted strings.

**Lemma 2:**
Gluing together strings with matching doubled $k$-mers yield unitigs.
Big picture

Input k-mers

Parallel partial compaction algorithm

Intermediate sequences

Parallel glue algorithm

Unitigs
Input sequences

Cannot load all sequences in memory. Need again to partition. Would like to have in the same partition.

Union-find of doubled kmers

Sequences of each U-F class are loaded and glued in parallel.
BCALM 2’s glue module

Input sequences

Cannot load all sequences in memory. Need again to partition.
Would like to have , , and in the same partition.

Sequences of each U-F class are loaded and glued in parallel.
20 Gbp spruce and 22 Gbp pine

Previously,
spruce: 2 days, 1380 cores, 4.3 TB RAM (Abyss)  [Birol 2013]
pine: 3 months, 32 cores, 0.8 TB RAM (MaSuRCA)  [Zimin 2014]

<table>
<thead>
<tr>
<th></th>
<th>Pine</th>
<th>Spruce</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>8 h 25 m</td>
<td>8 h 52 m</td>
</tr>
<tr>
<td>Memory</td>
<td>17 GB</td>
<td>31 GB</td>
</tr>
<tr>
<td>Unitigs</td>
<td>30.5 Gbp</td>
<td>56.0 Gbp</td>
</tr>
<tr>
<td>#</td>
<td>257 M</td>
<td>580 M</td>
</tr>
</tbody>
</table>

1.1/1.2 TB compressed reads

$k = 61$, abundance cut-off 7, 8/16 threads (pine/spruce)

$k$-mer counting time not included: 1 day, ≤ 40 GB memory, DSK 2
# Human dataset

<table>
<thead>
<tr>
<th>Human NA18507</th>
<th>Bcalm 2</th>
<th>Bcalm 1</th>
<th>ABySS-P 1.9</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Time</strong></td>
<td><strong>2 h</strong></td>
<td>13 h</td>
<td>6.5 h</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td><strong>2.8 GB</strong></td>
<td>43 MB</td>
<td>89 GB</td>
</tr>
</tbody>
</table>

54 GB compressed reads  
\(k = 55\), abundance cut-off 3, 16 threads  
\(k\)-mer counting time included in BCALM 1&2: 46 mins, 2 GB memory, DSK 2  
Meraculous: 16 hours, \(\leq 1\) TB  
[Georganas 2014]
Conclusion

Compacting de Bruijn graphs:
- efficient
  - 2 days for spruce, vs few CPU-years other methods
  - 2 hours for human
  - 2 GB memory per genome Gbp
- useful module for Illumina assemblers
- unitigs to replace $k$-mers in some applications

Observations:
- bottleneck becomes $k$-mer counting again
- not a data structure (construction algorithm, no queries)

Contact:
- @RayanChikhi, @pashadag, @NP_Malfoy on Twitter