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

Rayan Chikhi (CNRS)

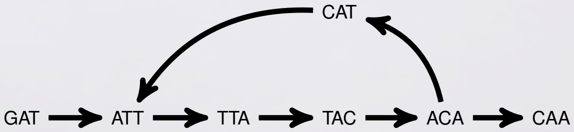
joint work with A. Limasset (ENS Rennes), P. Medvedev (Penn State)

ISMB 2016

de Bruijn Graph

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

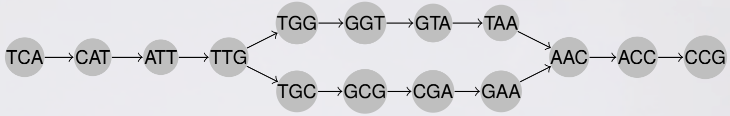
nodes: k-mers (words of length k)
edges: exact suffix-prefix overlaps of length $k-1$



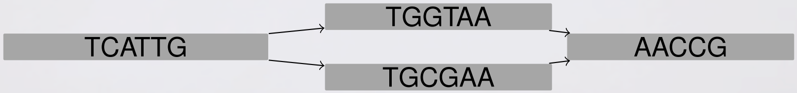
- assembly of genomes, metagenomes
- variant calling
- RNA-seq assembly & quantification

Compacted de Bruijn Graph

non-compacted de Bruijn graph:



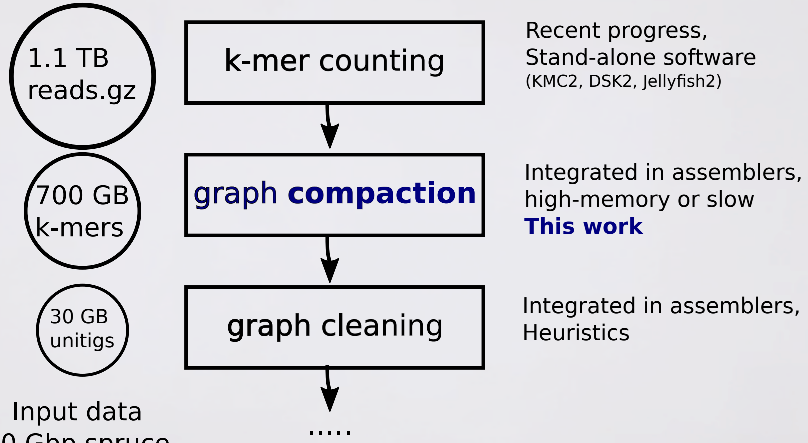
Compacted de Bruijn graph:



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
Recent progress, Stand-alone software (KMC2, DSK2, Jellyfish2)
700 GB k-mers
graph **compaction**
Integrated in assemblers, high-memory or slow
This work
30 GB unitigs
graph cleaning
Integrated in assemblers, Heuristics
.....

Input data
20 Gbp spruce
[Birol 2013]

- computationally intensive
- bottlenecks at early stages

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)

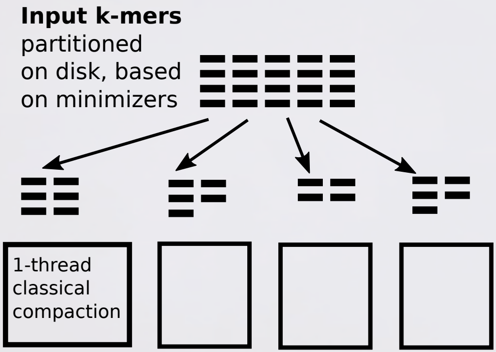
BCALM 2

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



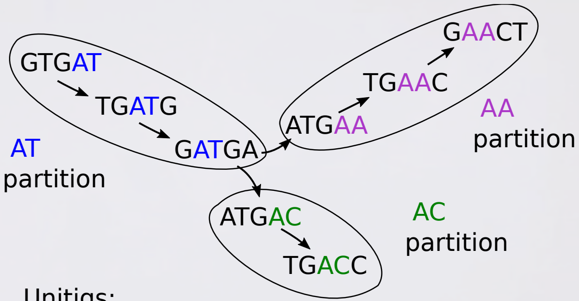
minimizer of *s*:
smallest ℓ-mer in *s*
[Roberts *et al*, 2004]

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

```
TGACGGG
 GACGGGT
  ACGGGTC
   CGGGTCA
    GGGTCAG
     GGTCAGA
```

Frequency ordering
$\rightarrow$ better repartition.
[RECOMB'14]

Compaction of partitions

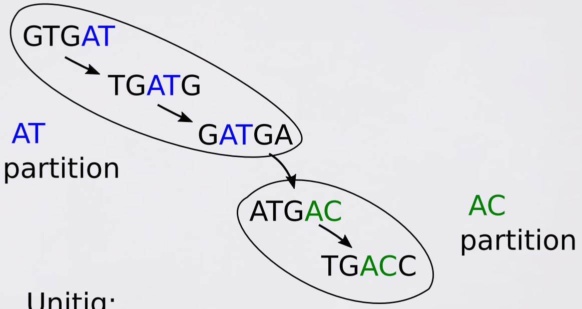


Unitigs:

GTGATGA
ATGACC
ATGAACT

k-mers are partitioned w.r.t minimizer.
In this case, compacting all partitions returns exactly all the unitigs.

Compaction of partitions (2)

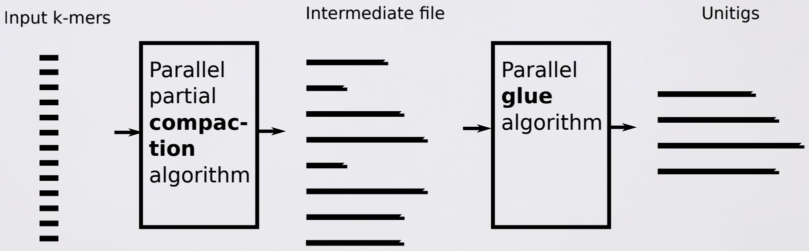


Unitig:

GTGATGACC

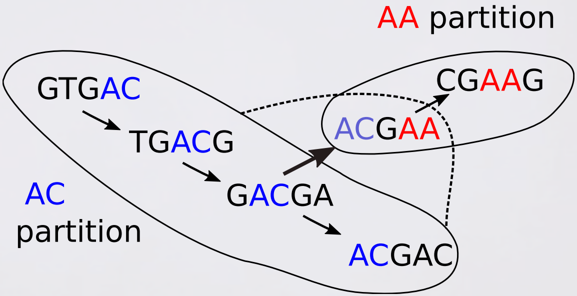
This case indicates that partitions contain sub-strings of unitigs.
Those substrings need to be later merged.

2-step strategy


Input k-mers
Parallel partial **compaction** algorithm
Intermediate file
Parallel **glue** algorithm
Unitigs

Simple partitioning is not enough

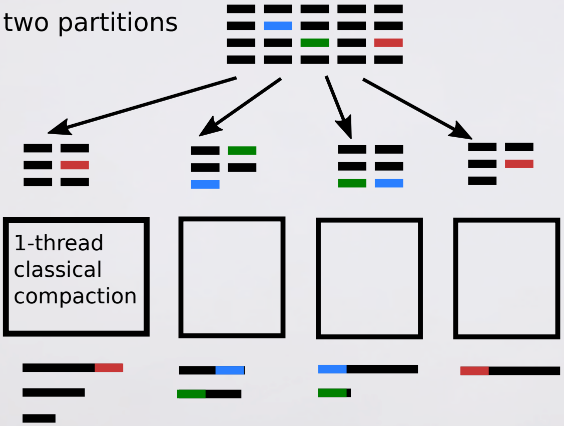
Compacting partitions may create false unitigs (due to missing edges).



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

BCALM 2’s partial compaction module

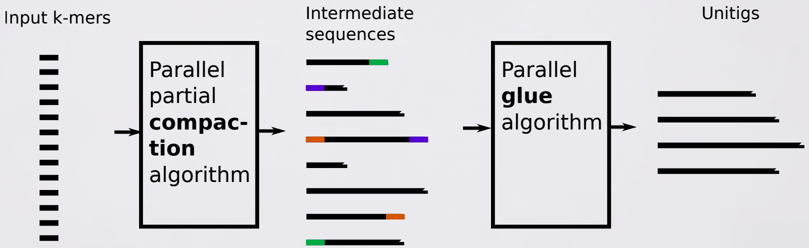
Doubled kmers are inserted in two partitions



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

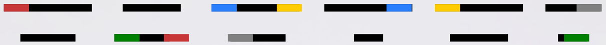
Lemma 2: Gluing together strings with matching doubled *k*-mers yield unitigs.

Big picture


Input k-mers
Parallel partial **compaction** algorithm
Intermediate sequences
Parallel **glue** algorithm
Unitigs

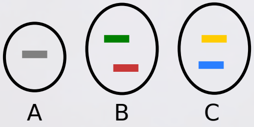
BCALM 2's glue module

Input sequences



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

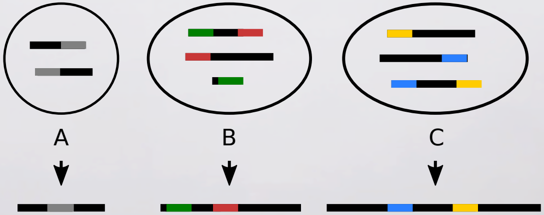
Union-find
of doubled kmers



Minimal perfect
hash table

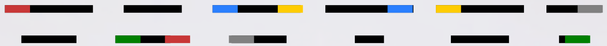
1	🟨	C
2	🟩	B
3	🟦	C
4	🟥	B
5	▨	A

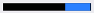
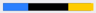
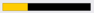
Sequences of
each U-F class
are loaded
and glued
in parallel.



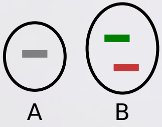
BCALM 2’s glue module

Input sequences



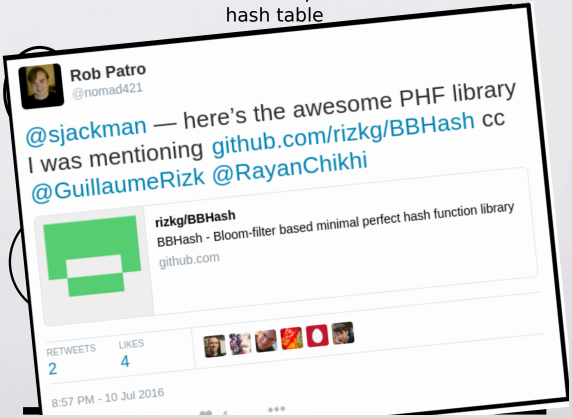
Cannot load all sequences in memory. Need again to partition.
Would like to have ,  and  in the same partition.

Union-find
of doubled kmers



Minimal perfect
hash table

Sequences of
each U-F class
are loaded
and glued
in parallel.



Rob Patro
@nomad421

@sjackman — here’s the awesome PHF library I was mentioning github.com/rizkg/BBHash cc @GuillaumeRizk @RayanChikhi

rizkg/BBHash
BBHash - Bloom-filter based minimal perfect hash function library
github.com

RETWEETS 2
LIKES 4

8:57 PM - 10 Jul 2016

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]

BCALM 2	Pine	Spruce
Time	8 h 25 m	8 h 52 m
Memory	17 GB	31 GB
Unitigs	30.5 Gbp	56.0 Gbp
#	257 M	580 M

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, $\leq$ 40 GB memory, DSK 2

Human dataset

Human NA18507	Bcalm 2	Bcalm 1	ABySS-P 1.9
Time	**2 h**	13 h	6.5 h
Memory	**2.8 GB**	43 MB	89 GB

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, ≤ 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