TwoPaCo

An efficient algorithm to build the compacted de Bruijn graph
from many complete genomes

Gaspare Ferraro

University of Pisa
Department of Computer Science

Pisa, 29 May 2018

Parte I

Introduction

A pan-genomic algorithm

De Bruijn graph

Compacted de Bruijn graph

The problem

Junctions

Parte II

The algorithm

Naive algorithm

- Store all $(k+1)$-mers in a hash table
- For each k-mers query the possible edge
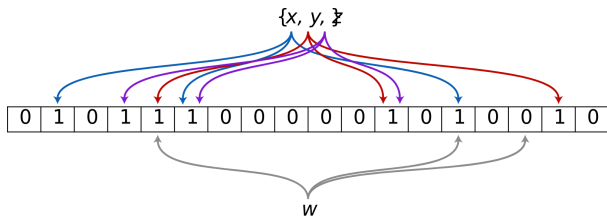- If multiple matching, mark as a junction

The memory issue

Bloom filter

A space-efficient probabilistic hash table

Bitmap V of size b, h hash functions $f_0, f_1, ..., f_{h-1} : U \rightarrow [0, b-1]$

$\text{insertion}(x) \rightarrow V[f_i(x)] = 1, \forall \ 0 \leq i < h$

$\text{contains}(x) \rightarrow$ **probabily yes** if $V[f_i(x)] == 1, \forall \ 0 \leq i < h$

Probability of false positive, after n insertion: $p_{FP} \simeq (1 - e^{-hn/m})^h$

Two Pass version

The memory issue[2]

Multiple rounds: dealing with memory restrictions

Parallelization scheme

Parte III

Results

Source code & Dataset

Original source code by medvedev group available on:

https://github.com/medvedevgroup/TwoPaCo

Personal implementation available on:

https://github.com/GaspareG/TwoPaCo

Dataset for experiments:

5 humans (from human reference genome)
8 primates (from ...)
62 Escherichia coli (from ...)
100 simulated human (from ...)

Memory complexity

The memory complexity is the maximum
among the first and the second pass of TwoPaCo

- First pass: insert all k-mers in a bloom filter of size b
- Second pass: store all **junction candidates** in a hash table

How many junction candidates?

- Real junction, J
- False positive induced from the bloom filter, FP

Result: $O(\max(b, (J + FP)k))$

Time complexity

The memory complexity is the sum
between the first and the second pass of TwoPaCo

- First pass: insert all k-mers in a bloom filter of size b using h hash functions
- Second pass: store all **junction candidates** in a hash table

How many k-mers?
$O(m)$, where $m = \Sigma_{s \in S}|s|$ is the total input size

How many junction candidates?

- Real junction, J
- False positive induced from the bloom filter, FP

Result: $O(mh + (|G_c| + FP)k)$

Performance comparison

Previous approach to the problem:

- Sibelia (St. Petersburg University, RU, 2012)
- SplitMEM (Stony Brook University, USA, 2014)
- bwt-based (Harbin Institute of Technology, CN, 2016)
- TwoPaCo (The Pennsylvania State University, USA, 2016)

Algorithm	Time complexity	Memory complexity
Sibelia	$O(m)$	$O(m)$
SplitMEM	$O(m \log g)$	$O(m + |G_c|)$
bwt-based	$O(m)$	$O(m)$
TwoPaCo	$O(mh + (|G_c| + FP)k)$	$O(\max(b, (J + FP)k))$

- $m = \Sigma_{s \in S} |s|$, total input size
- $g = \max_{s \in S} |s|$, size of the biggest genoma
- J and G_c, number of vertex and edge in the de Bruijn Graph
- b and h, size of the bloom filter table and number of hash functions
- FP, number of false positives in first pass

Fixed memory

How many round do we need to compress the graph
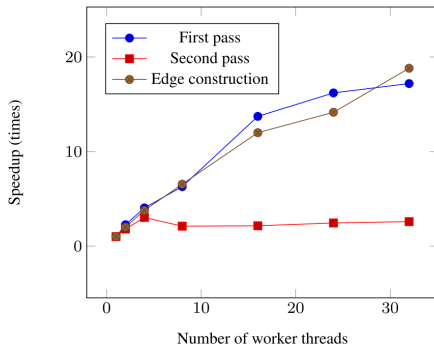without exceeding a given memory threshold?

Threshold	Used memory	Bloom filter size	Running time	Rounds
10GB	8.62GB	8.59GB (2^{33})	4h	1
8GB	6.73GB	4.29GB (2^{32})	7h	3
6GB	5.98GB	4.29GB (2^{32})	9h	4
4GB	3.51GB	2.14GB (2^{31})	11h	6
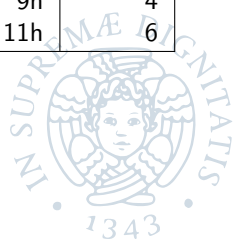
Parallel scalability

<center>How does the performance of TwoPaCo improve
to the increasing of working threads?</center>

Parallel scalability

- First pass: great improvement thanks to concurrent bloom filter
- Second pass: slight improvement due to race-conditions on the hash table
- Edge construction: great improvement thanks to k-mers independency

Bloom Filter false positive

Is the Bloom Filter really useful in reducing junction candidates?

Dataset	k	Initial	First pass	Second pass	%Fp
62 E.coli	25	$3.101 * 10^8$	$2.464 * 10^7$	$2.457 * 10^7$	0.31%
62 E.coli	100	$3.101 * 10^8$	$2.284 * 10^7$	$9.492 * 10^6$	58.45%
7 humans	25	$2.120 * 10^{10}$	$3.489 * 10^9$	$2.974 * 10^9$	14.78%
7 humans	100	$2.120 * 10^{10}$	$1.374 * 10^9$	$1.882 * 10^8$	86.30%
8 primates	25	$2.454 * 10^{10}$	$5.423 * 10^9$	$5.401 * 10^9$	0.39%
8 primates	100	$2.454 * 10^{10}$	$1.174 * 10^9$	$5.024 * 10^8$	57.20%

- Initial: total number of k-mers in dataset
- First pass: number of junction candidates (using a bloom filter)
- Second pass: real number of junction (using an hash table)
- False positive: 100*(First pass - Second pass)/(First pass)

Conclusion

Thanks for your attention!