

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



Junctions



The problem



Parte II

The algorithm



Naive algorithm

- Store all $(k + 1)$ -mers in a hash table
- For each k -mers query the possible edge
- If only 1 in and 1 out edge, unmark as a junction

Algorithm 1: FILTER-JUNCTIONS

Input : strings $S = \{s_1, \dots, s_m\}$ genome sequences

integer k , size of k -mers

Candidate set of junctions C_{in} (for now all positions are marked)

Output: A reduce candidate set of junctions C

```
1 foreach  $s \in S$  do
2   for  $1 \leq i < |s| - k$  do
3     if  $C[s, i] = \text{marked}$  then
4        $E \leftarrow E \cup s[i..i+k] \cup s[i-1..i+k-1]$     ▷ Store all  $(k+1)$ -mers

5 foreach  $s \in S$  do
6   for  $1 \leq i < |s| - k$  do
7     if  $C[s, i] = \text{marked}$  then
8        $(in, out) \leftarrow (0, 0)$     ▷ Count in/out edges
9       foreach  $c \in \{A, C, G, T\}$  do
10        if  $v \cdot c \in E$  then
11           $in \leftarrow in + 1$ 
12        if  $c \cdot v \in E$  then
13           $out \leftarrow out + 1$ 
14        if  $(in, out) = (1, 1)$  then
15           $C[s, i] = \text{unmarked}$     ▷ surely not a junction

16 return  $C$ 
```



The memory issue

First part of the naive algorithm:

```
foreach  $s \in S$  do  
  for  $1 \leq i < |s| - k$  do  
    if  $C[s, i] = \textit{marked}$  then  
       $E \leftarrow E \cup s[i..i + k] \cup s[i - 1..i + k - 1]$ 
```

We don't really need and, in almost all practical cases,
we can't store all the possible $(k + 1)$ -mers.

Mainly because **only a little percentual** of them are
junction in the de Bruijn graph.



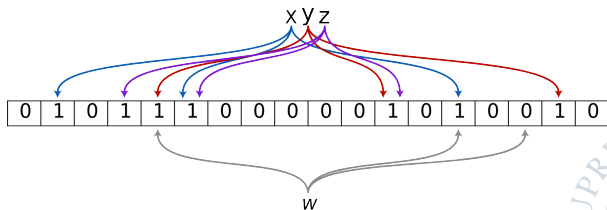
Bloom filter

A space-efficient probabilistic hash table

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

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

contains(x) \rightarrow **probably 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

- First pass: Select a set of junction candidates by insert all the $(k + 1)$ -mers in a bloom filter of choosing size
- Second pass: Filter out the false positive by storing the reduce sets of $(k + 1)$ -mers in an hash table

Algorithm 2: FILTER-JUNCTIONS-TWO-PASS

Input : strings $S = \{s_1, \dots, s_m\}$ genome sequences

integer k , size of k -mers

integer b , size of bloom filter

Candidate set of junctions C_{in}

Output: A reduce candidate set of junctions C_{out}

- 1 $F \leftarrow$ empty bloom filter of size b
 - 2 $C_{temp} \leftarrow \text{FILTER-JUNCTIONS}(S, k, F, C_{in})$
 - 3 $H \leftarrow$ empty hash table
 - 4 $C_{out} \leftarrow \text{FILTER-JUNCTIONS}(S, k, H, C_{in})$
 - 5 **return** C_{out}
-

▷ First pass

▷ Second pass



The memory issue²



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/GaspereG/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 $m = \sum_{s \in S} |s|$ is the total input size

How many junction candidates?

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

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



Complexity 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	$O(mh + (G_c + FP)k)$	$O(\max(b, (J + FP)k))$

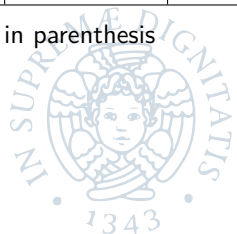
- $m = \sum_{s \in S} |s|$, total input size
- $g = \max_{s \in S} |s|$, size of the biggest genoma
- J and G_c , number of vertex and edge in the de Bruijn Graph
- b and h , size of the bloom filter table and number of hash functions
- FP , number of false positives in first pass

Running time comparison

Dataset	Sibelia	SplitMEM	bwt-based	TwoPaCo	
				1 thread	15 t
62 E.coli (k=25)	0 (12.2)	70 (178.0)	8 (0.85)	4 (0.16)	2 (0.16)
62 E.coli (k=100)	8 (7.6)	67 (178.0)	8 (0.50)	4 (0.19)	2 (0.19)
7 humans (k=25)	-	-	867 (100.30)	436 (4.40)	63 (4.40)
7 humans (k=100)	-	-	807 (46.02)	317 (8.42)	57 (8.42)
8 primates (k=25)	-	-	-	914 (34.36)	111 (34.36)
8 primates (k=100)	-	-	-	756 (56.06)	101 (56.06)
50 humans (k=25)	-	-	-	-	705 (56.06)
50 humans (k=100)	-	-	-	-	927 (56.06)

Running times in minutes and memory usage in gigabytes in parenthesis

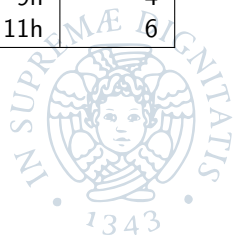
- - on Sibelia = out of time
- - on SplitMEM = out of memory
- - on bwt-based = out of memory



Fixed memory

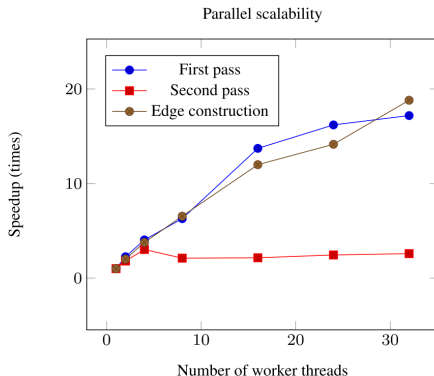
How many rounds do we need to compress the graph without exceeding a given memory threshold?

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

How does the performance of TwoPaCo improve to the increasing of working threads?



- 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: $100 * (\text{First pass} - \text{Second pass}) / (\text{First pass})$



Conclusion

Thanks for your attention!

