

SGA

Stringing the genome together

BME 235, April 23

Josh Espinoza, Chris Kan, Audrey Musselman-Brown

SGA - Overview

- Memory Efficient
- CPU Intensive for initial steps
- Modular approach
 - Pipeline benefits
 - Computational benefits
- Emphasis on Substring Accuracy

How does SGA Compare?

Table 1. Assembly statistics for *C. elegans* data set

	SGA	Velvet	ABYSS	SOAPdenovo
Scaffold N50 size	26.3 kbp	31.3 kbp	23.8 kbp	31.1 kbp
Aligned contig N50 size	16.8 kbp	13.6 kbp	18.4 kbp	16.0 kbp
Mean aligned contig size	4.9 kbp	5.3 kbp	6.0 kbp	5.6 kbp
Sum aligned contig size	96.8 Mbp	95.2 Mbp	98.3 Mbp	95.4 Mbp
Reference bases covered	96.2 Mbp	94.8 Mbp	95.9 Mbp	95.1 Mbp
Reference bases covered by contigs ≥ 1 kb	93.0 Mbp	92.1 Mbp	93.9 Mbp	92.3 Mbp
Mismatch rate at all assembled bases	1 per 21,545 bp	1 per 8786 bp	1 per 5577 bp	1 per 26,585 bp
Mismatch rate at bases covered by all assemblies	1 per 82,573 bp	1 per 18,012 bp	1 per 8209 bp	1 per 81,025 bp
Contigs with split/bad alignment (sum size)	458 (4.4 Mbp)	787 (7.2 Mbp)	638 (9.1 Mbp)	483 (4.4 Mbp)
Total CPU time	41 h	2 h	5 h	13 h
Max memory usage	4.5 GB	23.0 GB	14.1 GB	38.8 GB

How does SGA Compare?

Table 1. Assembly statistics for *C. elegans* data set

	SGA	Velvet	ABYSS	SOAPdenovo
Scaffold N50 size	26.3 kbp	31.3 kbp	23.8 kbp	31.1 kbp
Aligned contig N50 size	16.8 kbp	13.6 kbp	18.4 kbp	16.0 kbp
Mean aligned contig size	4.9 kbp	5.3 kbp	6.0 kbp	5.6 kbp
Sum aligned contig size	96.8 Mbp	95.2 Mbp	98.3 Mbp	95.4 Mbp
Reference bases covered	96.2 Mbp	94.8 Mbp	95.9 Mbp	95.1 Mbp
Reference bases covered by contigs ≥ 1 kb	93.0 Mbp	92.1 Mbp	93.9 Mbp	92.3 Mbp
Mismatch rate at all assembled bases	1 per 21,545 bp	1 per 8786 bp	1 per 5577 bp	1 per 26,585 bp
Mismatch rate at bases covered by all assemblies	1 per 82,573 bp	1 per 18,012 bp	1 per 8209 bp	1 per 81,025 bp
Contigs with split/bad alignment (sum size)	458 (4.4 Mbp)	787 (7.2 Mbp)	638 (9.1 Mbp)	483 (4.4 Mbp)
Total CPU time	41 h	2 h	5 h	13 h
Max memory usage	4.5 GB	23.0 GB	14.1 GB	38.8 GB

How long might this take?

Stage	Processes	Wall time	CPU time	Max Memory
Build index (raw)	123	23 hr	187 hr	45 GB
Correct reads	63	1 hr	355 hr	28 GB
Build index (corrected)	123	32 hr	355 hr	44 GB
Filter reads	1	33 hr	167 hr	54 GB
Merge reads	1	15 hr	105 hr	48 GB
Assemble reads	3	23 hr	41 hr	16 GB
Align to contigs	62	6 hr	210 hr	10 GB
Build scaffolds	4	7 hr	7 hr	13 GB
All stages	-	140 hr	1427 hr	54 GB

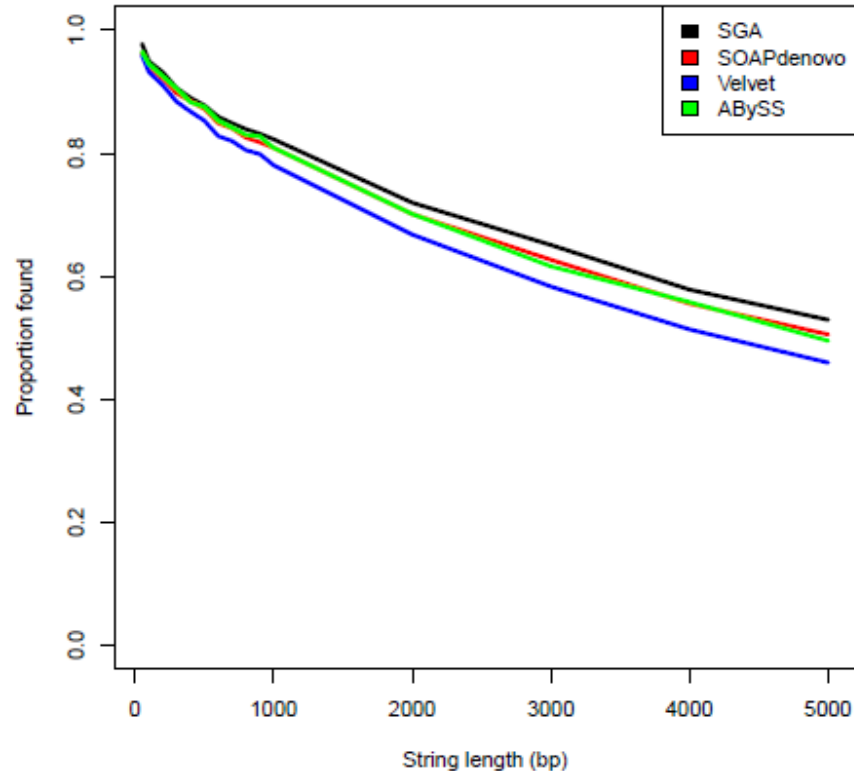
Table 3.3: Running time and memory summary for the SGA human genome assembly

How long might this take?

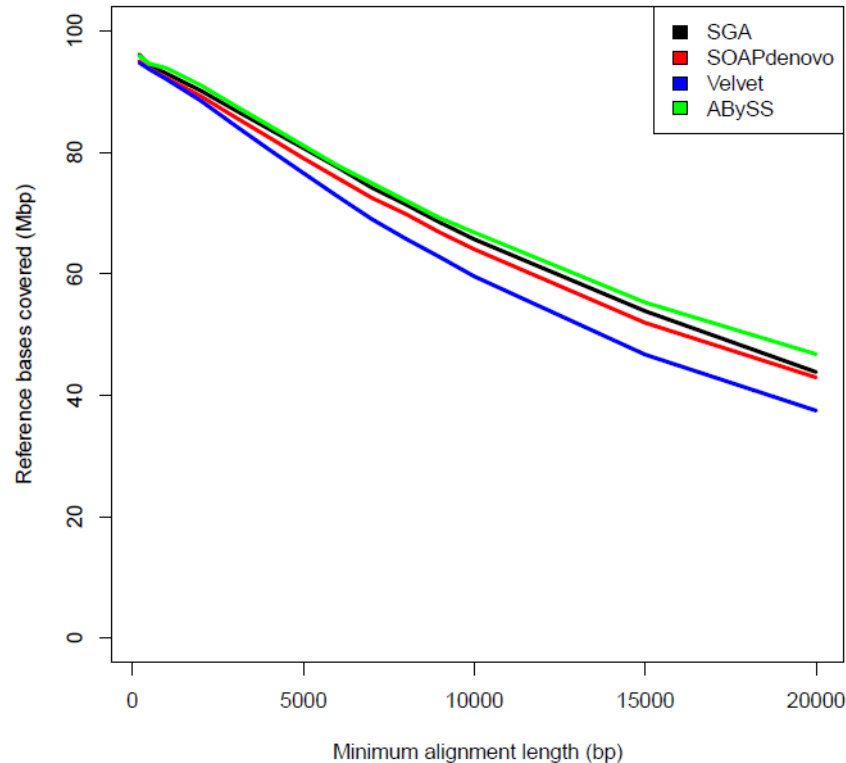
Stage	Processes	Wall time	CPU time	Max Memory
Build index (raw)	123	23 hr	187 hr	45 GB
Correct reads	63	1 hr	355 hr	28 GB
Build index (corrected)	123	32 hr	355 hr	44 GB
Filter reads	1	33 hr	167 hr	54 GB
Merge reads	1	15 hr	105 hr	48 GB
Assemble reads	3	23 hr	41 hr	16 GB
Align to contigs	62	6 hr	210 hr	10 GB
Build scaffolds	4	7 hr	7 hr	13 GB
All stages	-	140 hr	1427 hr	54 GB

Correction and Indexing take up 63% of CPU time.

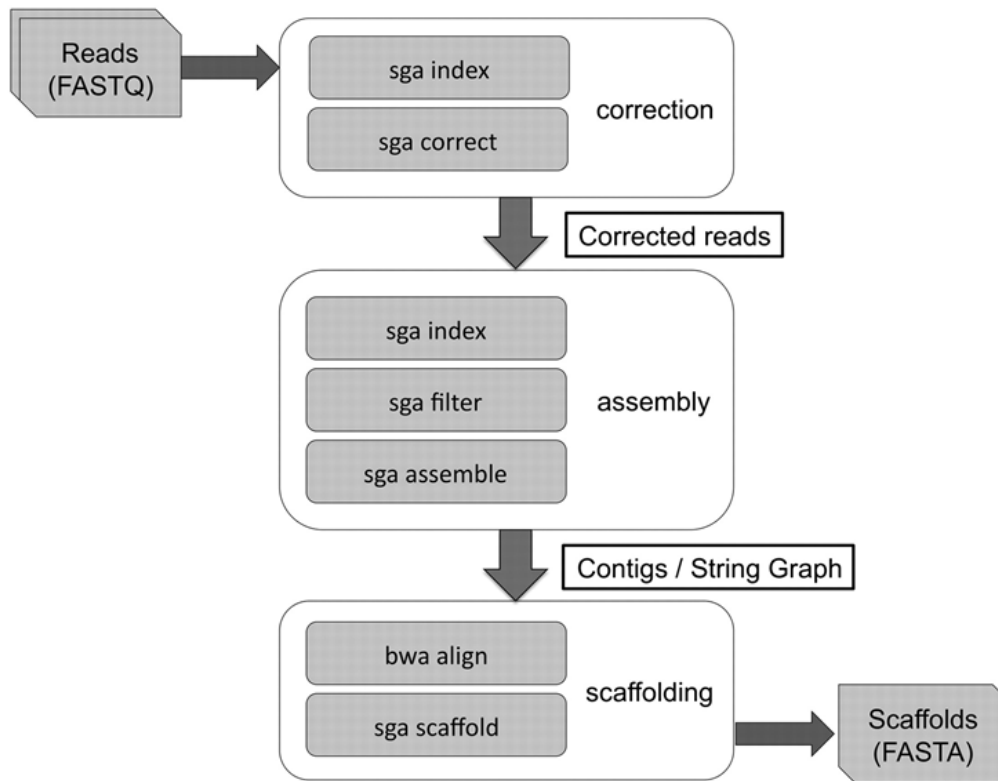
Accuracy - Substring Coverage



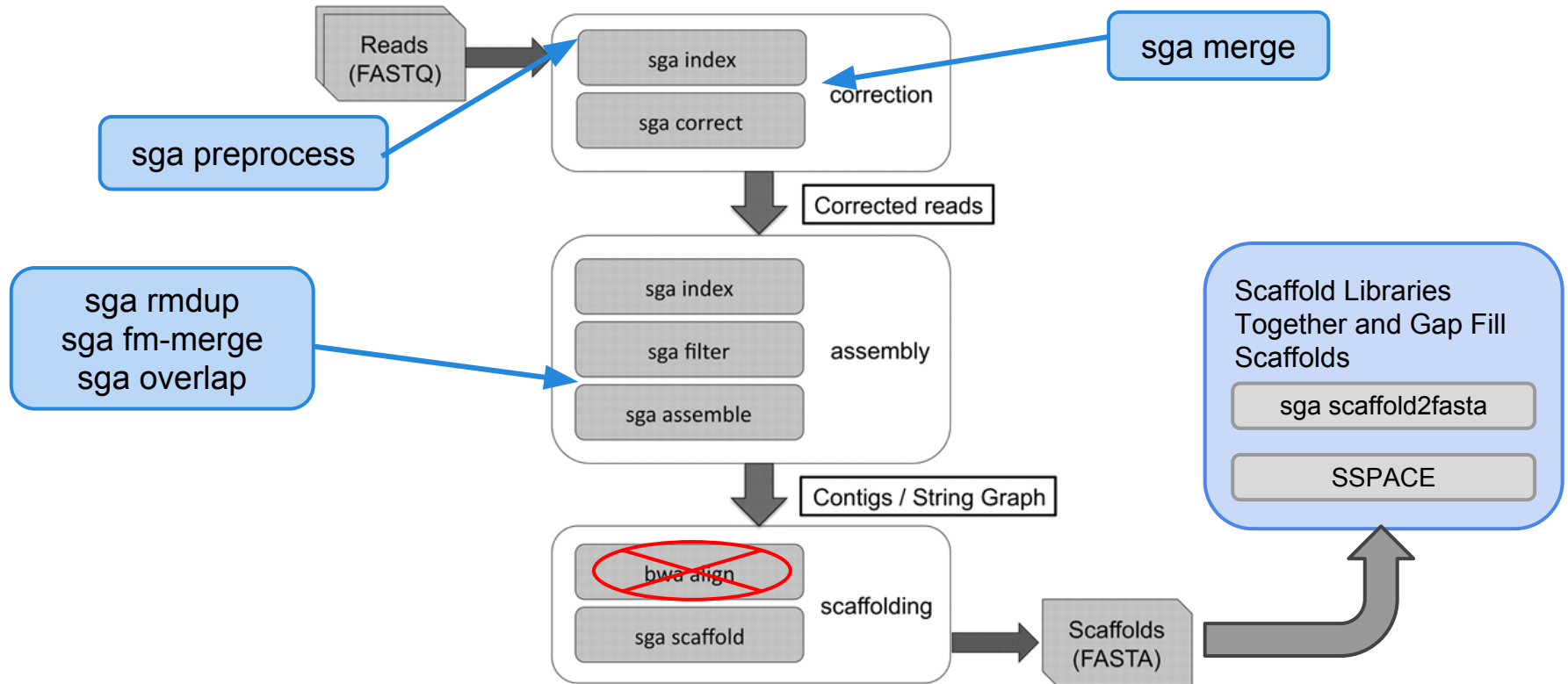
Accuracy - Reference Coverage



The SGA Pipeline



The SGA Pipeline



Installation Steps

i) Install Google-sparsehash

```
$ wget http://sparsehash.googlecode.com/files/sparsehash-2.0.2.tar.gz
$ gtar xvzf sparsehash-1.8.1.tar.gz
$ cd sparsehash-1.8.1
$ ./configure --prefix=`pwd`
$ make
$ make install
```

ii) Install bamtools

```
$ git clone git://github.com/pezmaster31/bamtools.git
$ cd bamtools
$ mkdir build
$ cd build
$ cmake ..
$ make
```

iii) Install sga

```
$ git clone https://github.com/jts/sga.git
$ cd sga/src
$ ./autogen.sh
$ ./configure --prefix=/path/to/sga_dir/ --with-sparsehash=/path/to/sparsharsh_dir/ --with-bamtools=/path/to/bamtools_dir/
$ make
$ make install
```

SPECIAL THANKS TO ROBERT
CALEF FOR THE CS WISDOM

Requirements

Depth of coverage

Minimum = 20 - 30X coverage

Recommended = 40X coverage

Read lengths

Designed for 100 nt reads or greater

Overlap

Too short : Edges will be created in graph between reads that have short repeats at their ends

Too long : No overlap

Recommended : For 100 nt reads : 65 nt for FM-merge, and 75 for SGA assemble

Parameter Tuning

- the overlap size (-m to sga overlap/sga assemble)
- the k-mer size used in error correction
- the k-mer threshold in error correction (sga correct -x)

<https://github.com/jts/sga/wiki/Parameter-tuning>

Burrows Wheeler Transformation [BWT]

- Reversible compression technique
- Rearranges string into runs of similar chars
- Linear time with respect to the $\text{len}(p)$: $O(|P|)$ time, independent of the $\text{len}(S)$ (*where P is pattern and S is string*)

Algorithm Basics

1. Create suffix array
2. Sort all rotations into lexicographic order
3. Store first | last columns (F and L, respectively)

BWT | Algorithm eg.

Query string:

BANANA

+

\$

\$ B A N A N A

A \$ B A N A N

N A \$ B A N A

A N A \$ B A N

N A N A \$ B A

A N A N A \$ B

B A N A N A \$

Step 1

\$ B A N A N A

A \$ B A N A N

A N A \$ B A N

A N A N A \$ B

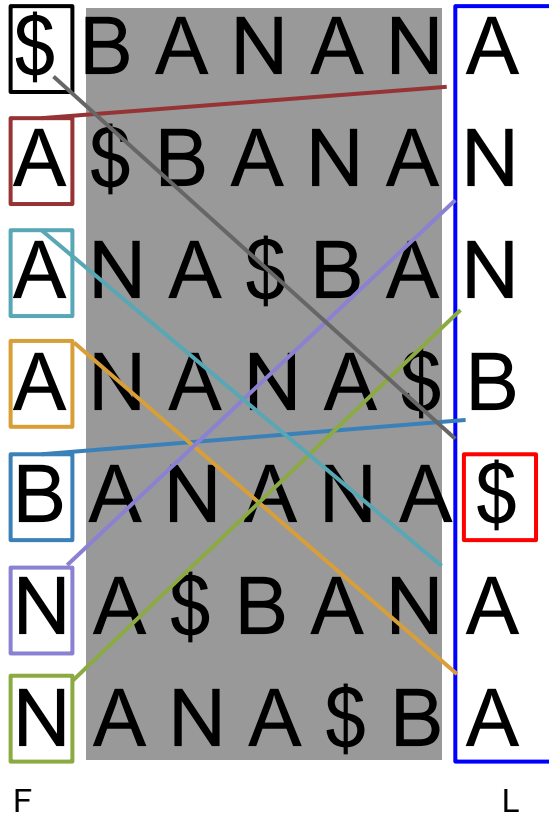
B A N A N A \$

N A \$ B A N A

N A N A \$ B A

Step 2

BWT | Algorithm eg. cont.



- Step 3**
- i) Store last column of sorted substrings
 - ii) Find substring with last char == \$ (s0)
 - iii) Since last substring's 1st char == 1st B; go to string in last col with 1st B (s1) : out += B
 - iv) s1[0] = 3rd A; find substring whose last char == 3rd A (s2) : out += A
 - v) s2[0] = 2nd N == s3[-1] : out += N
 - vi) s3[0] = 2nd A == s4[-1] : out += A
 - vii) s4[0] == 1st N == s5[-1] : out += N
 - viii) s5[0] = 1st A == s6[-1] : out += A
 - ix) s6[0] = \$ == s7[-1] : out += \$

B A N A N A \$

FM-Index

- Full-text **index** in **Minute** space.
- Compressed full-text **substring index** based on the **Burrows-Wheeler transform** and Suffix Array
- Scales with the size of the input alphabet Δ
- Locating each pattern occurrence takes **$O(\log |\Delta| (\log^2 n / \log \log n))$** time. *<http://people.unipmn.it/manzini/papers/spire04.pdf>*

FM Index | Algorithm eg.

Query string = 'abaaba' : How to find occurrences of substring 'aba' in string?

Step 1) Create sorted suffix array

Step 2) Store F and L columns (BWT)

Step 3) Store some offsets of suffix array [SA] based on sample-rate

F		L	SA	
\$	a b a a b a		6	\$
a	\$ a b a a b		5	a \$
a	a b a \$ a b		2	a a b a \$
a	b a \$ a b a		3	a b a \$
a	b a a b a \$		0	a b a a b a \$
b	a \$ a b a a		4	b a \$
b	a a b a \$ a		1	b a a b a \$

Offsets: 0, 3

FM Index | Algorithm eg.

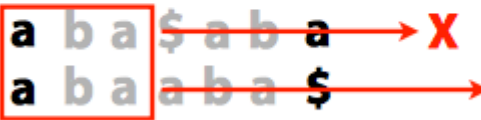
Query string = 'abaaba' : How to find occurrences of substring 'aba' in string?

Step 4) Determine range of BWT rows that have substring prefix

Step 5) Lookup for row 4, entry of SA present

Step 6) Lookup for row 3, entry of SA discarded

<i>F</i>	<i>L</i>	<i>SA</i>	Rows
\$	a b a a b a	6	0
a	\$ a b a a b		1
a	a b a \$ a b	2	2
a	b a \$ a b a		3
a	b a a b a \$	0	4
b	a \$ a b a a	4	5
b	a a b a \$ a		6



FM-Index | Algorithm eg.

Query string = 'abaaba' : How to find occurrences of substring 'aba' in string?

Step 6a) LF map tells us that **a** at the beginning of R2 is the same **a** at the end of R3

Step 6b) Row 2 has SA = 2

Step 6c) $2(\text{R2's SA}) + 1$ (# step from R2 >> R3)
 $(2) + (1) = 3$
R3 has SA = 3

<i>F</i>		<i>L</i>	<i>SA</i>	Row				
\$	a	b	a	a	b	a	6	0
a	\$	a	b	a	a	b		1
a	a	b	a	\$	a	b	2	2
a	b	a	\$	a	b	a		3
a	b	a	a	b	a	\$	0	4
b	a	\$	a	b	a	a	4	5
b	a	a	b	a	\$	a		6

FM-Index | Parameters

Components of the FM Index:

First column $[F]$: $\sim |\Delta|$ integers (First column of BWT)

Last column $[L]$: m characters (Last column of BWT)

SA Sample : $m * a$ integers, where a is fraction of rows kept

Checkpoints: $m * |\Delta| * b$ integers, where b is sample-rate

FM-Index | eg.

i) DNA = {A,T,C,G} 2 bit/nt; ii) T = human genome; iii) $a = 1/32$; iv) $b = 1/128$

First Column [F] : 16 bytes

Last Column [L] : 2 bits * $3.0E9$ chars = 750 MB

SA Sample : $3.0E9$ chars * (4 bytes/char)/32 = ~ 400 MB

Checkpoints : $3.0E9 * (4 \text{ bytes/char}) * (1/128) = \sim 100$ MB

Total < 1.5 GB

FM-Index | Algorithm eg. cont.

- At the expense of adding some SA values ($O(m)$ integers) to index, can be done using much less memory
- Small memory footprint!

Assembly Algorithm

- FM Index
- Merge paths to reduce graph size
- Reindex
- Build string graph



Read Merging

- Query FM index
- Follow unambiguous paths (unipaths)
- Merge them into a single read
- build FM index for new reads
- Remove graph tips (hairiness)



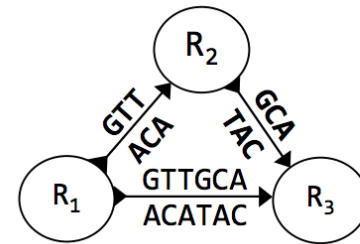
String Graph

- Modified overlap graph
- Remove all duplicate or contained reads
- Index with l-mers
- only reads sharing an l-mer are checked for an overlap

A

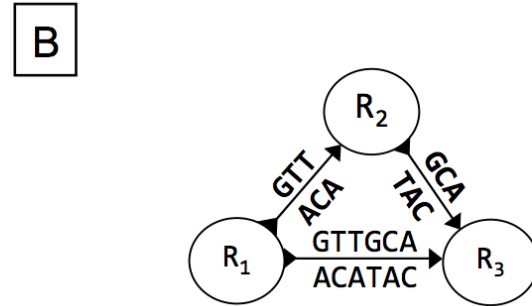
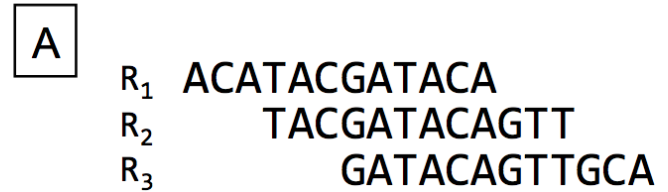
R₁ ACATACGATACA
R₂ TACGATACAGTT
R₃ GATACAGTTGCA

B



String Graph

- Label edges with non-matching sequence
- Remove transitive edges



Bubble Popping

- Similar to deBruijn graph assemblers
- Find pairs of nodes with multiple walks between them
- Choose a walk to stay in the graph
- If other walks are similar enough (default 95%), they are removed and set aside for heterozygosity analysis

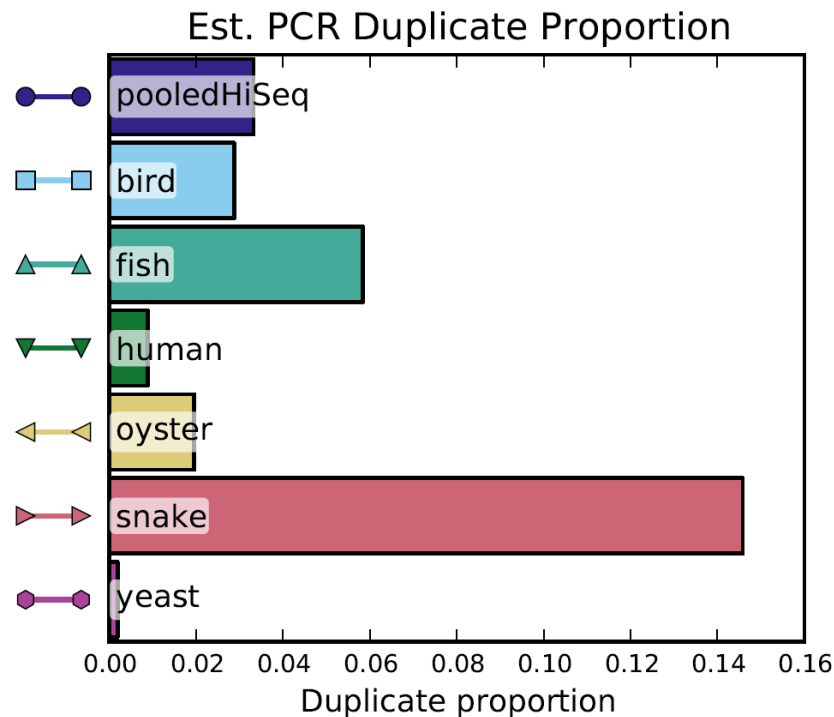
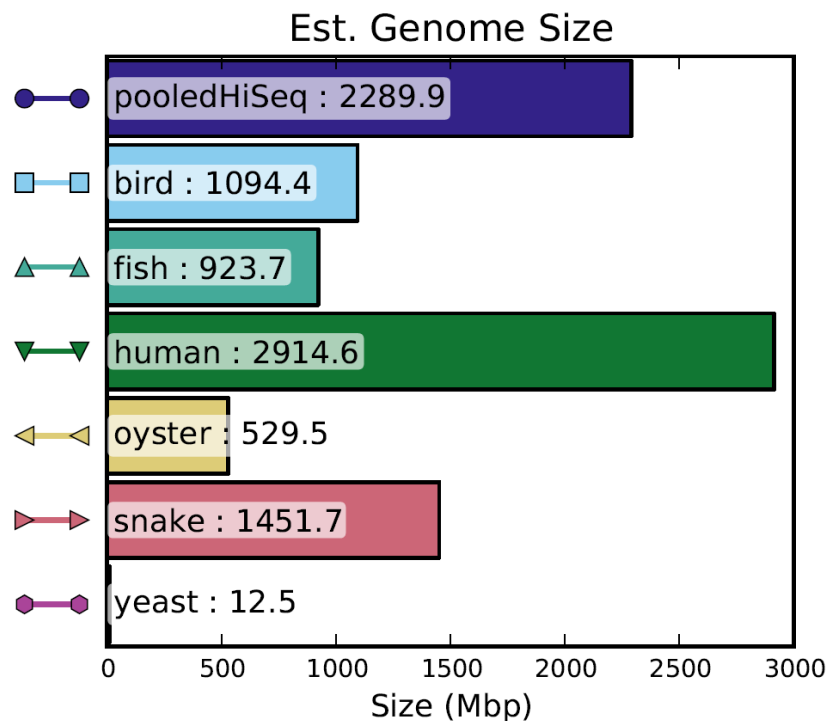
Scaffolding - Building the Scaffold

- Create potential order of contigs
- Determine unique or repetitive (A-Statistic)
- Remove repetitive or contigs of uncertain order

Scaffolding - Filling Scaffold Gaps

- Standalone
- Using kmer search on gap regions to find paths

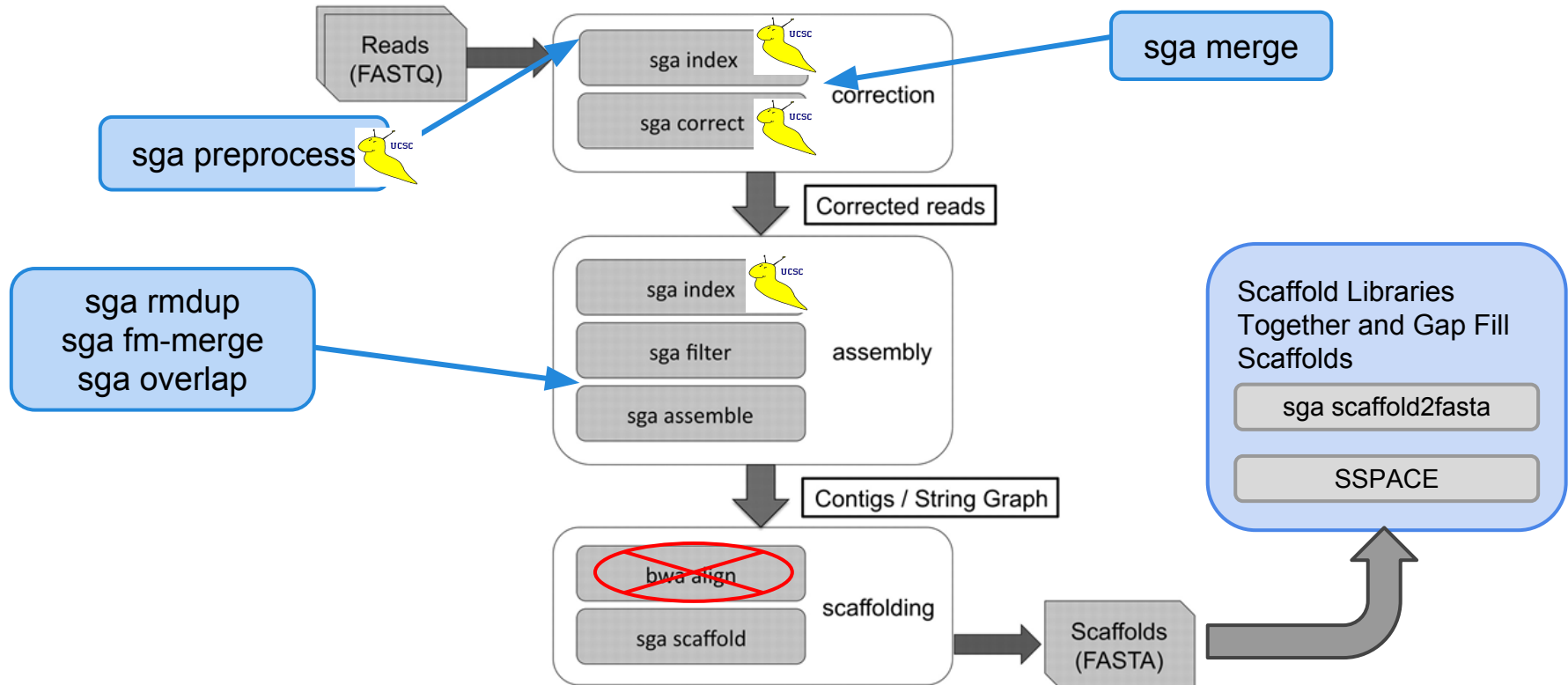
Fast QC/ Pre QC



The Road Forward

- Adjust Error Correction and Assembly
- Trying to assemble/ fine tune one library
- Final assembly of all libraries and merging

The Road Forward



References

1. Simpson and Durbin, “Efficient de Novo Assembly of Large Genomes Using Compressed Data Structures.”
2. Jared Thomas Simpson, “Efficient Sequence Assembly and Variant Calling Using Compressed Data Structures.”

