Illumina Assembly

Martin Norling

Uppsala, November 15th 2016

Sequencing recap

- This lecture is focused on illumina, but the techniques are the same for all short-read sequencers.
- Short reads are (generally) high quality and highly cost efficient.

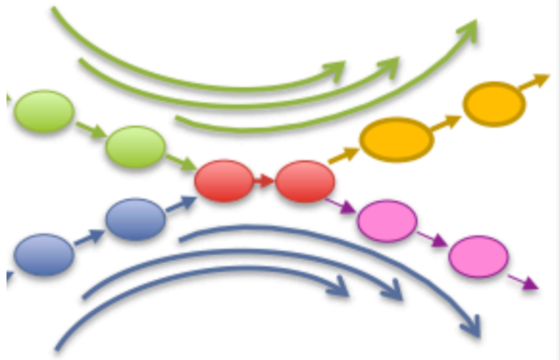
Why is it hard to assemble genomes?

Good quality assembly demands:

- High quality samples
- Well prepared sequencing libraries
- Good sequencing runs
- Sequence aware processing and assembly
- Correct result interpretation

What do we need?

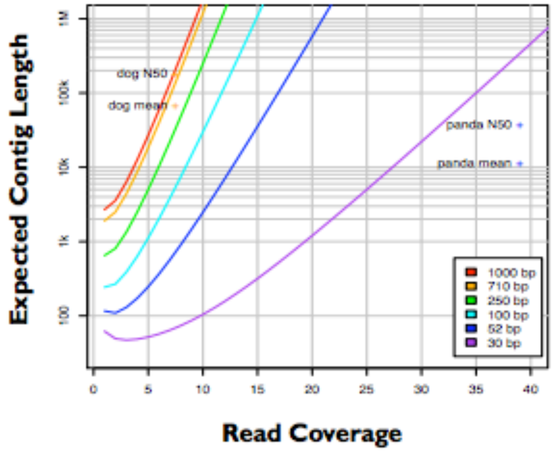
Read Length



Reads & mates must be longer than the repeats

- Short reads will have ***false overlaps*** forming hairball assembly graphs
- With long enough reads, assemble entire chromosomes into contigs

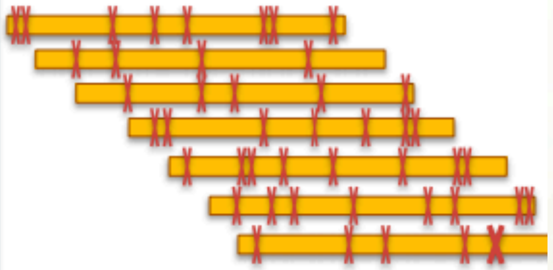
Coverage



High coverage is required

- Oversample the genome to ensure every base is sequenced with long overlaps between reads
- Biased coverage will also fragment assembly

Quality



Errors obscure overlaps

- Reads are assembled by finding kmers shared in pair of reads
- High error rate requires very short seeds, increasing complexity and forming assembly hairballs

Assembly strategies

Most people can come up with some strategy to assemble reads into sequences, but coming up with an effective and efficient strategy is difficult.

We will look at two of the most common strategies:

- Overlap, Layout, Consensus (commonly *OLC*)
- De Bruijn Graph based (sometimes *DBG*)

Random Reads

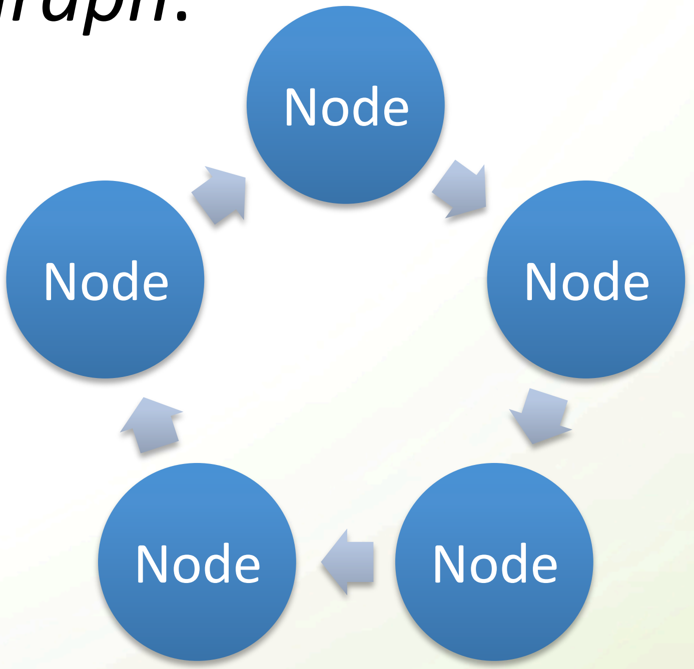
```
ACAGTGGCTGGGCGGATGACCCGACCTCTATGTCGTTGCCCGGCCCCTATCGAAGGCGAGTCATGAAGATGCACACGTTGTGTCCCACTACTGAACCCTC

 CAGTGGCTGGG  GATGACCCGAC TCTATGTCGTT CCCGGCCCCTA  GAAGGCGAGTC TGAAGATGCAC  GTTGTGTCCCA TACTGAACCCT
 CAGTGGCTGGG    TGACCCGACCT TATGTCGTTGC CGGCCCCTATC AAGGCGAGTCA GAAGATGCACA  TTGTGTCCCAC ACTGAACCCTC
    TGGCTGGGCGG      CGACCTCTATG  GTTGCCCGGCC  TATCGAAGGCG GTCATGAAGAT CACACGTTGTG  CCACTACTGAA
     GGCTGGGCGGA     CGACCTCTATG    TGCCCGGCCCC ATCGAAGGCGA TCATGAAGATG       TGTGTCCCACT
     GGCTGGGCGGA      GACCTCTATGT   TGCCCGGCCCC ATCGAAGGCGA  CATGAAGATGC      TGTGTCCCACT
     GGCTGGGCGGA      GACCTCTATGT   TGCCCGGCCCC  TCGAAGGCGAG CATGAAGATGC        TGTCCCACTAC
     GGCTGGGCGGA       ACCTCTATGTC   GCCCGGCCCCT   GAAGGCGAGTC  GAAGATGCACA       TCCCACTACTG
      GCTGGGCGGAT      ACCTCTATGTC   GCCCGGCCCCT     AGGCGAGTCAT AAGATGCACAC      TCCCACTACTG
          GGCGGATGACC  ACCTCTATGTC   GCCCGGCCCCT       GCGAGTCATGA  ATGCACACGTT   TCCCACTACTG
           GCGGATGACCC ACCTCTATGTC    CCCGGCCCCTA         AGTCATGAAGA GCACACGTTGT   CCACTACTGAA
            CGGATGACCCG  CTCTATGTCGT  CCCGGCCCCTA             ATGAAGATGCA             ACTACTGAACC
                       ACCTCTATGTC     CCGGCCCCTAT              GAAGATGCACA           ACTACTGAACC
                         CTCTATGTCGT     GGCCCCTATCG             AAGATGCACAC           CTACTGAACCC
                         CTCTATGTCGT      GCCCCTATCGA             AGATGCACACG          CTACTGAACCC
                         CTCTATGTCGT       CCCCTATCGAA            AGATGCACACG          CTACTGAACCC
                          TCTATGTCGTT      CCCCTATCGAA                 CACACGTTGTG       ACTGAACCCTC
                          TCTATGTCGTT      CCCCTATCGAA                                   ACTGAACCCTC
                           CTATGTCGTTG       CCTATCGAAGG
                           CTATGTCGTTG
                           CTATGTCGTTG
                             ATGTCGTTGCC
                              TGTCGTTGCCC
                               GTCGTTGCCCG
```

Graphs!

To get a long sequence out of short sequences they're piled up into a *graph*.

A graph is basically a set of *nodes* (in our case sequence reads) connected by *edges*.



Directed, *cyclic* graph with 5 *nodes* (vertices) and 5 *edges*

Overlap Layout Consensus (OLC)

This is the "naive" way of doing assembly, but also a very good way of doing assembly if the data allows it!

Algorithm has three stages:

1. **Overlap** – Find overlaps between reads
2. **Layout** – Collapse overlap graph into *contigs*
3. **Consensus** – Find consensus sequence for each contig

Overlap

The basic idea is to find all overlaps between all reads, and creating a graph. This operation is *extremely* costly.

There are optimizations:

- Suffix trees
- Indexes

But OLC is still always computationally expensive.

Layout

The graph from find all read overlaps can be extremely complex, so first the graph is reduced. There are different ways of doing this but commonly:

- Edges are removed if they can be inferred from other edges
- Edges with low support are assumed to be sequencing errors and removed.

Consensus

The final part is quite straight forward; try to find the most likely base for each position based on the graph.

OLC

Pros: Utilizes long reads well – fewer, longer reads are less expensive to overlap, and OLC can make use of the entire long reads.

Cons: Time consuming and requires large amounts of memory.

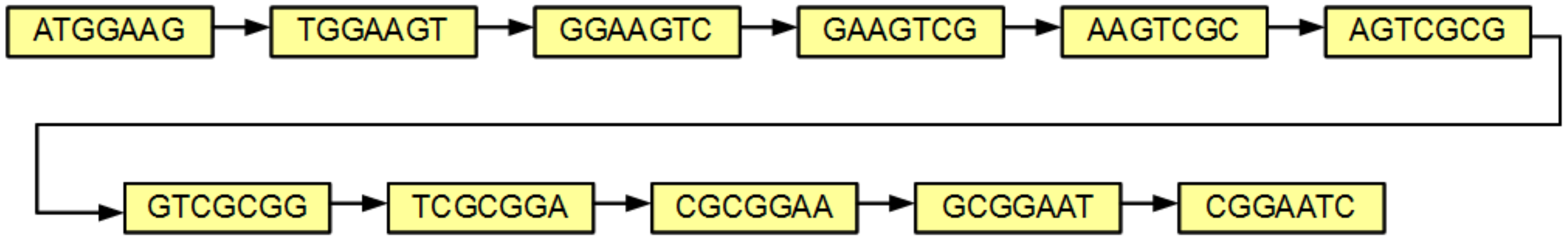
De Bruijn Graph based assembly

sequence **ATGGAAGTCGCGGAATC**

7mers

```
ATGGAAG
 TGGAAGT
  GGAAGTC
   GAAGTCG
    AAGTCGC
     AGTCGCG
      GTCGCGG
       TCGCGGA
        CGCGGAA
         GCGGAAT
          CGGAATC
```

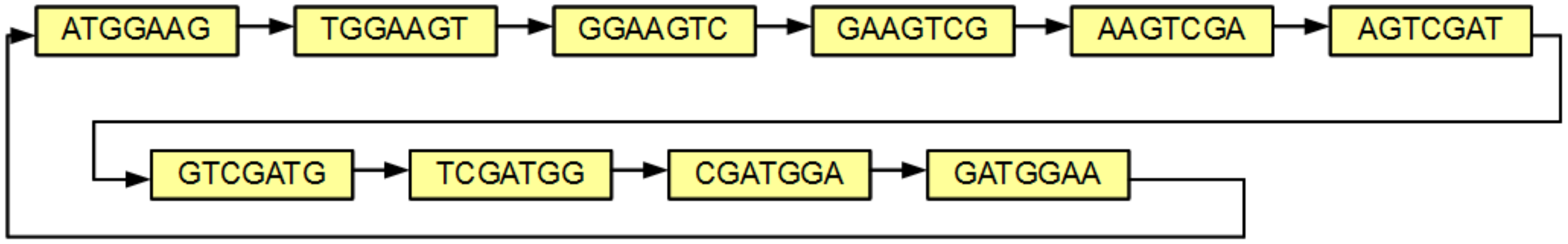
de Bruijn graph



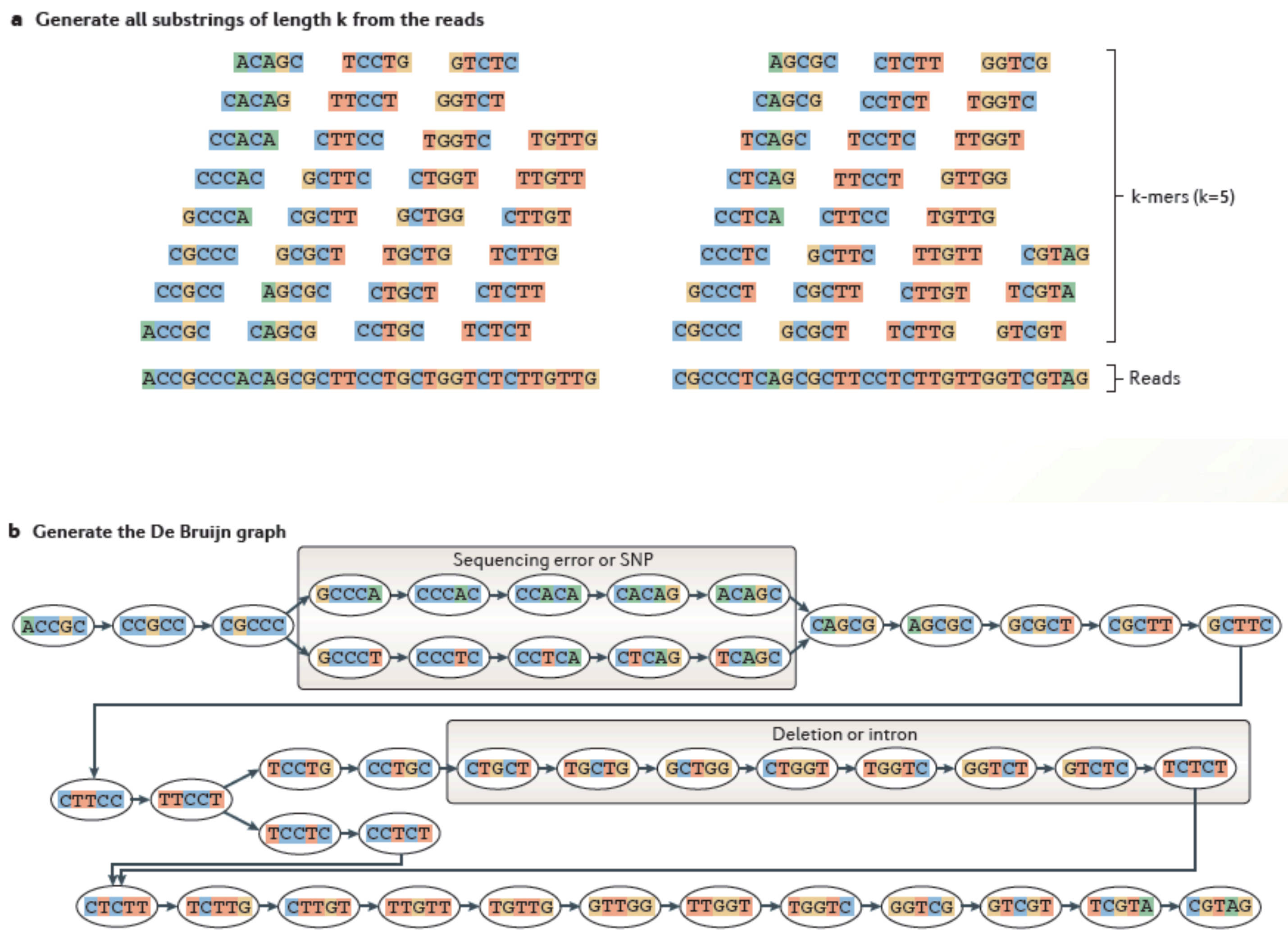
De Bruijn graph construction

ATGGAAGTCGATGGAAG

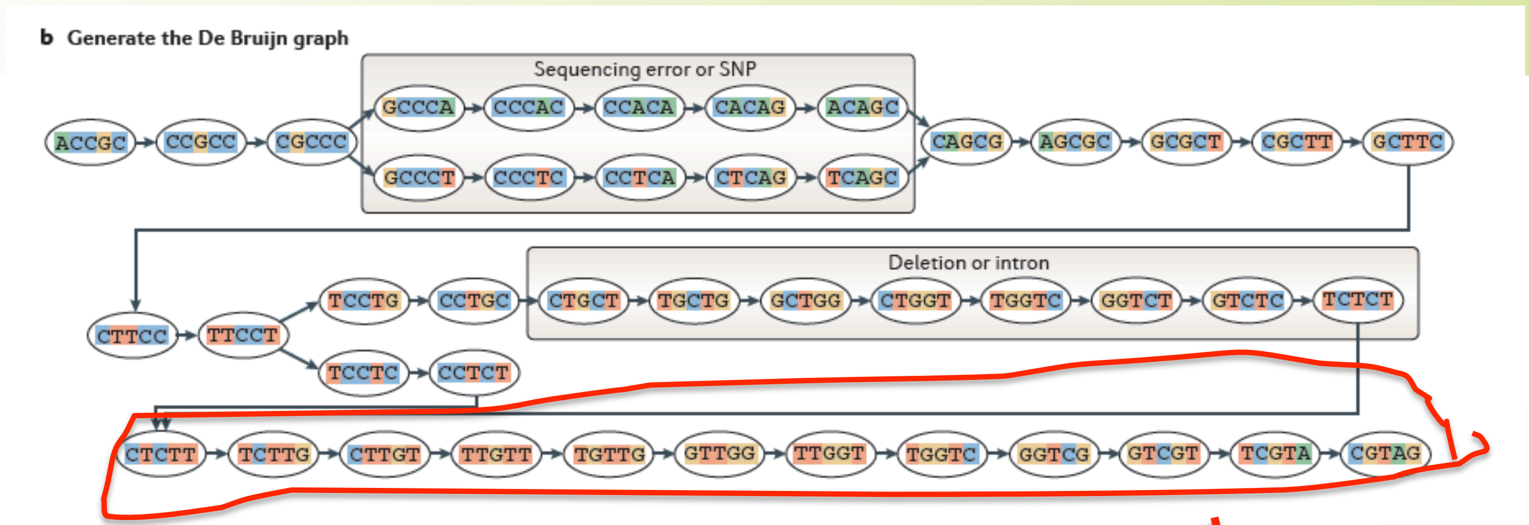
```
ATGGAAG
 TGGAAGT
  GGAAGTC
   GAAGTCG
    AAGTCGA
     AGTCGAT
      GTCGATG
       TCGATGG
        CGATGGA
         GATGGAA
          ATGGAAG
```

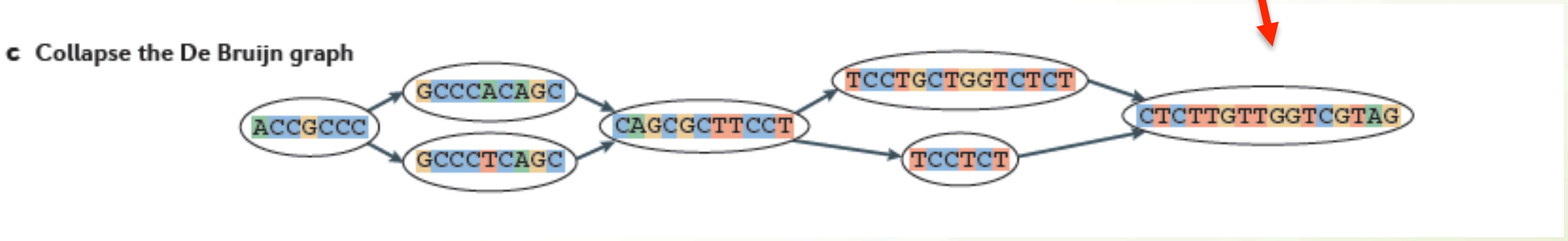


Sequence Assembly via De Bruijn Graphs



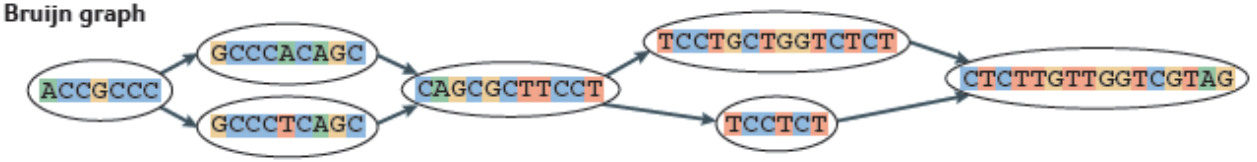
From Martin & Wang, Nat. Rev. Genet. 2011



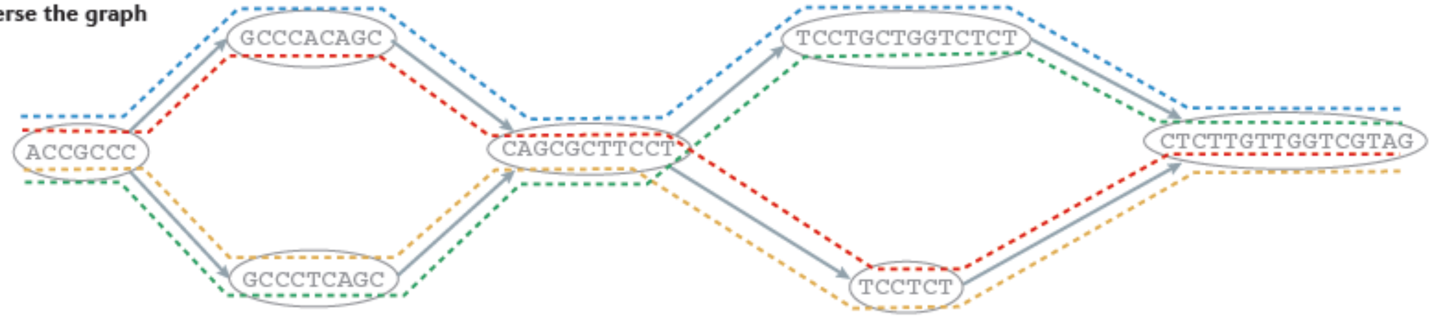


From Martin & Wang, Nat. Rev. Genet. 2011

c Collapse the De Bruijn graph



d Traverse the graph



e Assembled isoforms

🟦 ------	ACCGCCCACAGCGCTTCCTGCTGGTCTCTTGTTGGTCGTAG
🟥 ------	ACCGCCCACAGCGCTTCCT--------CTTGTTGGTCGTAG
🟨 ------	ACCGCCCTCAGCGCTTCCT--------CTTGTTGGTCGTAG
🟩 ------	ACCGCCCTCAGCGCTTCCTGCTGGTCTCTTGTTGGTCGTAG

From Martin & Wang, Nat. Rev. Genet. 2011

De Bruijn

- Pros: Computationally efficient, can work with large coverage short read datasets
- Cons: Sensitive to sequence errors, connection between assembly and read is lost, does not work so well with longer reads

Assemblathon 2

- Uses 454, Illumina, and PacBio for three large eukaryote genomes: a bird, a fish, and a snake
- Bird - Illumina 14 libraries, 454, PacBio
- Fish - Illumina, 8 libraries
- Snake - Illumina, 4 libraries
- Teams take the data, perform assemblies with whatever tools they wish, and then submit their results => teams are evaluated more than individual programs!

GigaScience 2013, **2**:10

Assemblathon 2

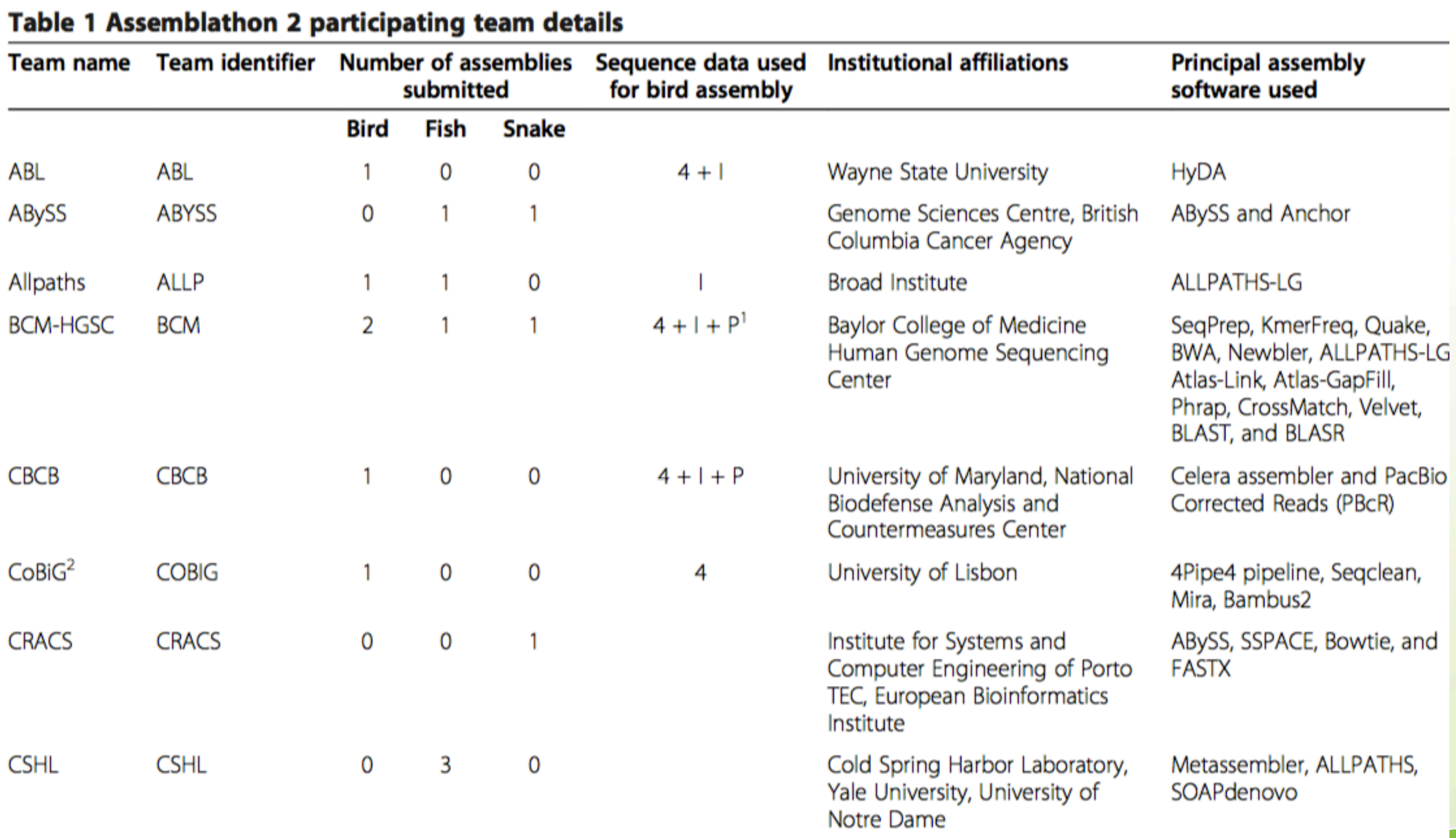
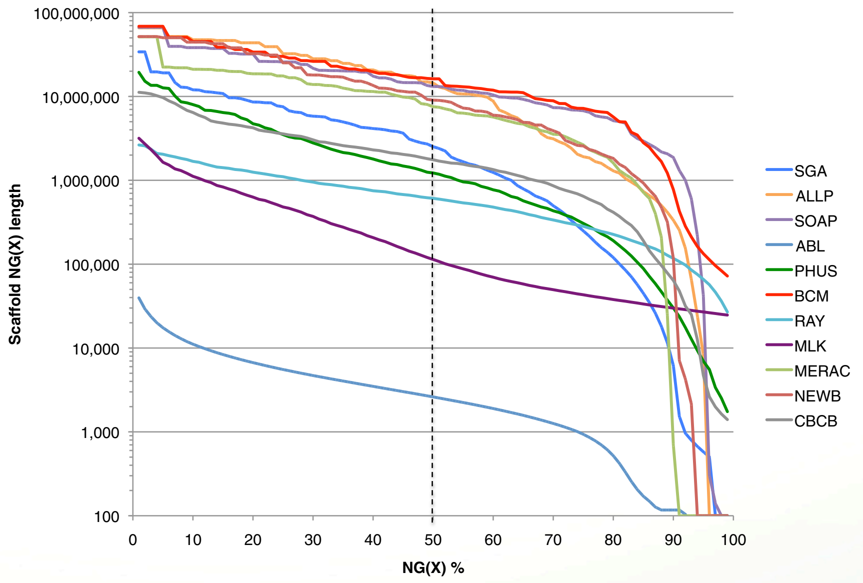
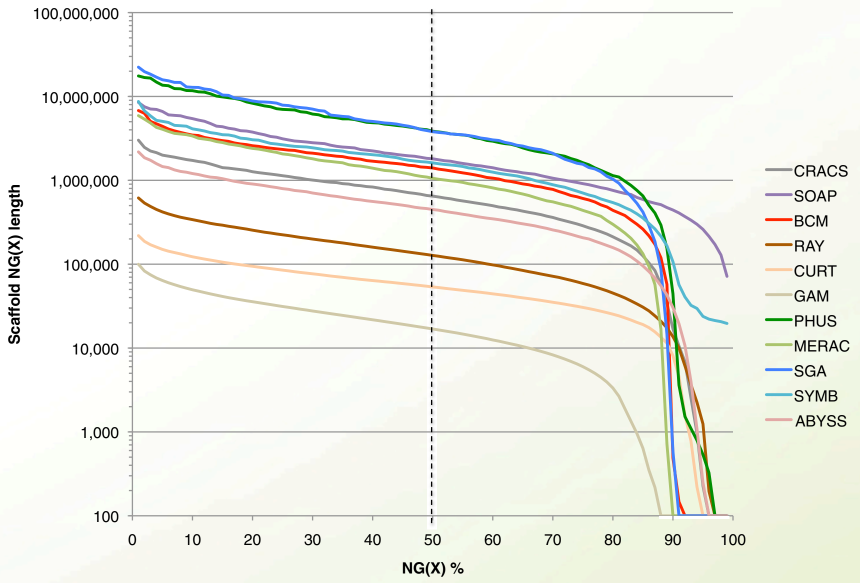


Table 1 Assemblathon 2 participating team details

Team name	Team identifier	Number of assemblies submitted			Sequence data used for bird assembly	Institutional affiliations	Principal assembly software used
		Bird	Fish	Snake			
ABL	ABL	1	0	0	4 + I	Wayne State University	HyDA
ABySS	ABYSS	0	1	1		Genome Sciences Centre, British Columbia Cancer Agency	ABySS and Anchor
Allpaths	ALLP	1	1	0	I	Broad Institute	ALLPATHS-LG
BCM-HGSC	BCM	2	1	1	4 + I + P[1]	Baylor College of Medicine Human Genome Sequencing Center	SeqPrep, KmerFreq, Quake, BWA, Newbler, ALLPATHS-LG Atlas-Link, Atlas-GapFill, Phrap, CrossMatch, Velvet, BLAST, and BLASR
CBCB	CBCB	1	0	0	4 + I + P	University of Maryland, National Biodefense Analysis and Countermeasures Center	Celera assembler and PacBio Corrected Reads (PBcR)
CoBiG[2]	COBIG	1	0	0	4	University of Lisbon	4Pipe4 pipeline, Seqclean, Mira, Bambus2
CRACS	CRACS	0	0	1		Institute for Systems and Computer Engineering of Porto TEC, European Bioinformatics Institute	ABySS, SSPACE, Bowtie, and FASTX
CSHL	CSHL	0	3	0		Cold Spring Harbor Laboratory, Yale University, University of Notre Dame	Metassembler, ALLPATHS, SOAPdenovo

Assemblathon 2 - Bird vs. Snake





Assemblathon 2 recommendations

- Based on the findings of Assemblathon 2, we make a few broad suggestions to someone looking to perform a *de novo* assembly of a large eukaryotic genome:
- 1. Don't trust the results of a single assembly. If possible, generate several assemblies (with different assemblers and/or different assembler parameters). Some of the best assemblies entered for Assemblathon 2 were the evaluation assemblies rather than the competition entries.
- 2. Do not place too much faith in a single metric. It is unlikely that we would have considered SGA to have produced the highest ranked snake assembly if we had only considered a single metric.
- 3. Potentially choose an assembler that excels in the area you are interested in (e.g., coverage, continuity, or number of error free bases).
- 4. If you are interested in generating a genome assembly for the purpose of genic analysis (e.g., training a gene finder, studying codon usage bias, looking for intron-specific motifs), then it may not be necessary to be concerned by low N50/NG50 values or by a small assembly size.
- 5. Assess the levels of heterozygosity in your target genome before you assemble (or sequence) it and set your expectations accordingly.

Some Assembly ~~Problems~~ Programs

There are way more assembly programs than algorithms, so if they use the same algorithm, why do they produce different results?

There are of course tons of tweaks and heuristics that make assemblers differ quite a lot from each other.

Here are some examples of common assemblers and how they work!

ABySS

ABySS – “Assembly By Short Sequences” – is a relatively basic de bruijn graph based assembler, with a strong focus on parallelization.

The assembler has two steps; (1) de Bruijn graph contig contruction and (2) contig joining with paired-end/mate-pair information.

Errors are handled by iterative removal of short “dead-end” branches and removal of small bubbles.

AllPaths-LG

ALLPATHS-LG (Large Genome) is a de Bruijn assembler specially tuned for handling large genomes, and as such it requires at least one mate-pair library and one paired-end library for assembly.

AllPaths does error correction on reads based on kmer abundances, is highly memory efficient to allow large assemblies, and adaptive to better handle low coverage regions.

MaSuRCA

MaSuRCA uses de Bruijn graphs to create unique extensions of all reads into what they call *super reads.* These reads are then assembled by OLC, as the *super read* construction (ideally) creates a data set of hundredfold fewer, longer reads than the original data.

SOAPdenovo2

SOAPdenovo2 uses “sparse” de Bruijn graphs by using a method similar to the super reads from MaSuRCA, as well as multiple kmer sizes in order to allow faster and more memory efficient graph construction.

Uses paired/mate-pair information in a second step to join and scaffold contigs.

SPAdes

SPAdes uses another de Bruijn graph variant. It creates a *multisized de Bruijn graph* using several kmer sizes. This graph is then directly manipulated using paired information into a *paired assembly graph*. This graph is then collapsed into contigs.

… and MANY more

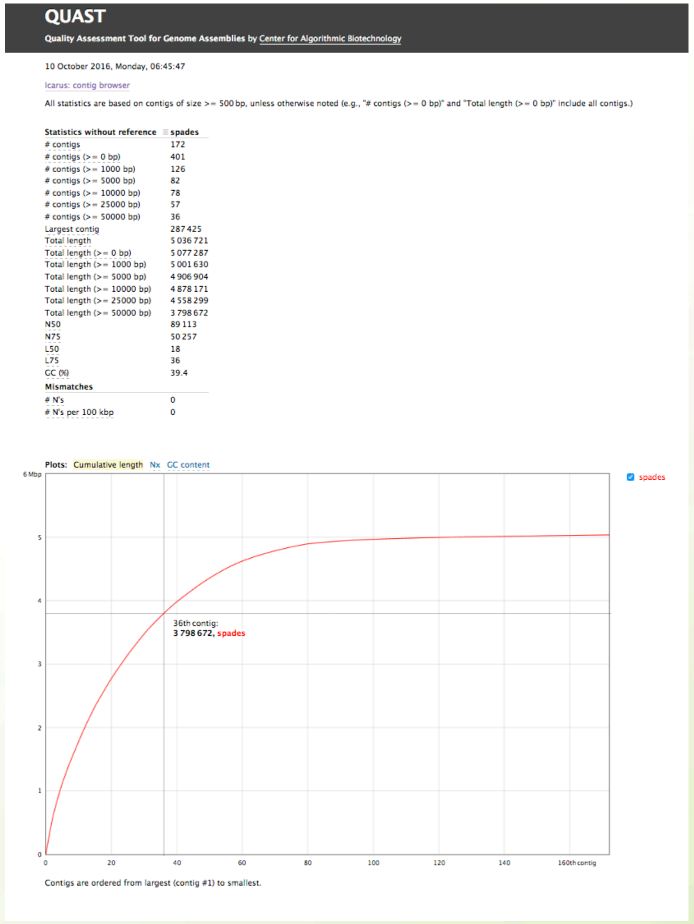
Name	Algorithm	Data
Abyss	De Bruijn	Illumina
Allpaths-lg	De Bruijn	Illumina/PacBio
CABOG (Celera)	OLC	All
Falcon	OLC	PacBio
HGAP	OLC	PacBio
Masurca	De Bruijn/OLC	All
Mira	“OLC”	All
Newbler	OLC	454/Illumina/Torrent
SGA	String	Illumina
SoapDeNovo	De Bruijn	Illumina
Spades	De Bruijn	Illumina (PacBio)

In short – there is endless ways to implement these algorithms

A first look at assemblies: QUAST

- QUAST, **Q**uality **A**ssessment **T**ool for **G**enome **A**ssemblies

Produces a basic report of common statistics, such as N50, number of contigs, etc.



QUAST

Quality Assessment Tool for Genome Assemblies by Center for Algorithmic Biotechnology

10 October 2016, Monday, 06:45:47

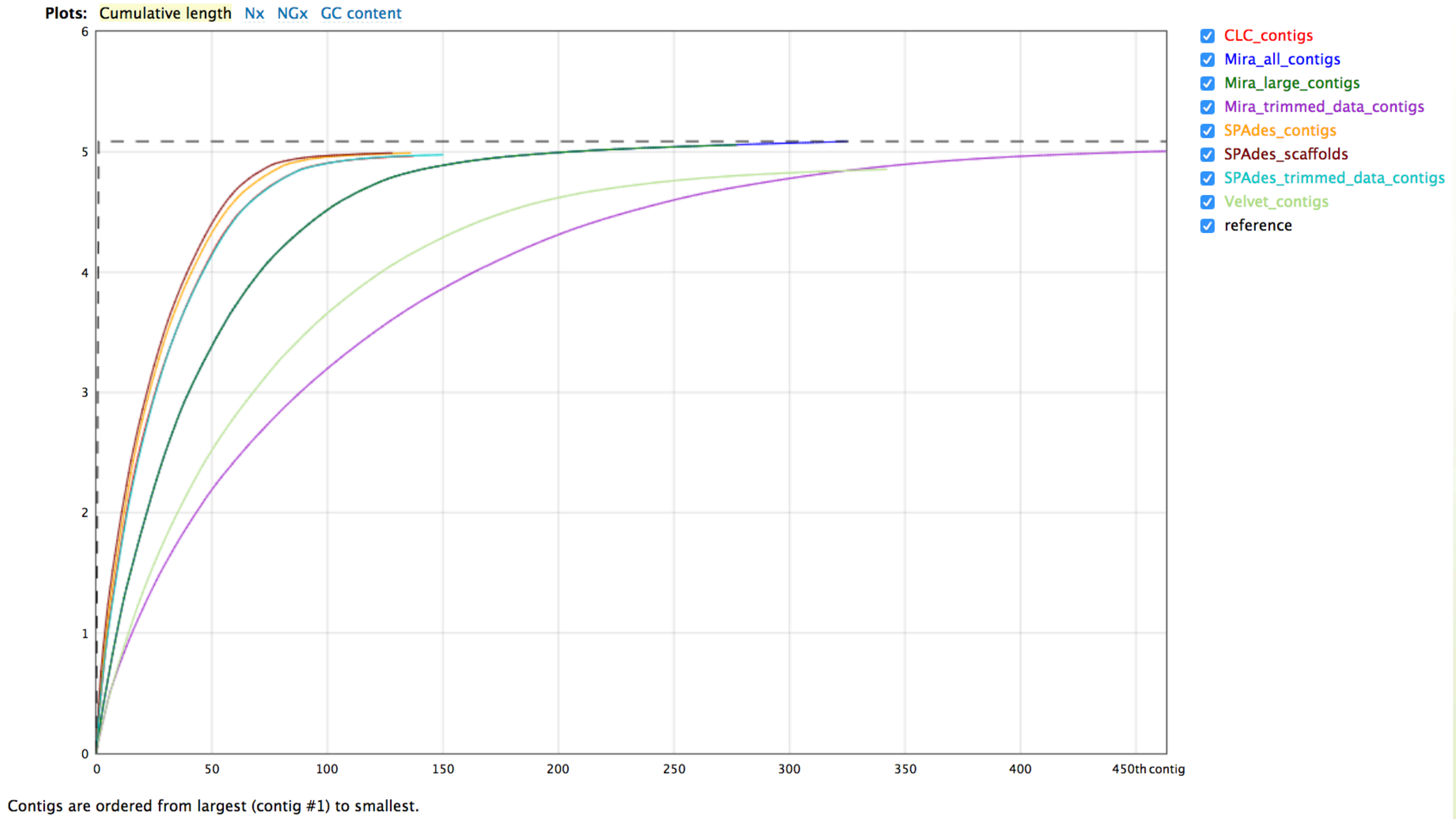
Icarus: contig browser

All statistics are based on contigs of size >= 500 bp, unless otherwise noted (e.g., "# contigs (>= 0 bp)" and "Total length (>= 0 bp)" include all contigs.)

Statistics without reference	spades
# contigs	172
# contigs (>= 0 bp)	401
# contigs (>= 1000 bp)	126
# contigs (>= 5000 bp)	82
# contigs (>= 10000 bp)	78
# contigs (>= 25000 bp)	57
# contigs (>= 50000 bp)	36
Largest contig	287 425
Total length	5 036 721
Total length (>= 0 bp)	5 077 287
Total length (>= 1000 bp)	5 001 630
Total length (>= 5000 bp)	4 906 904
Total length (>= 10000 bp)	4 878 171
Total length (>= 25000 bp)	4 558 299
Total length (>= 50000 bp)	3 798 672
N50	89 113
N75	50 257
L50	18
L75	36
GC (%)	39.4
Mismatches	
# N's	0
# N's per 100 kbp	0

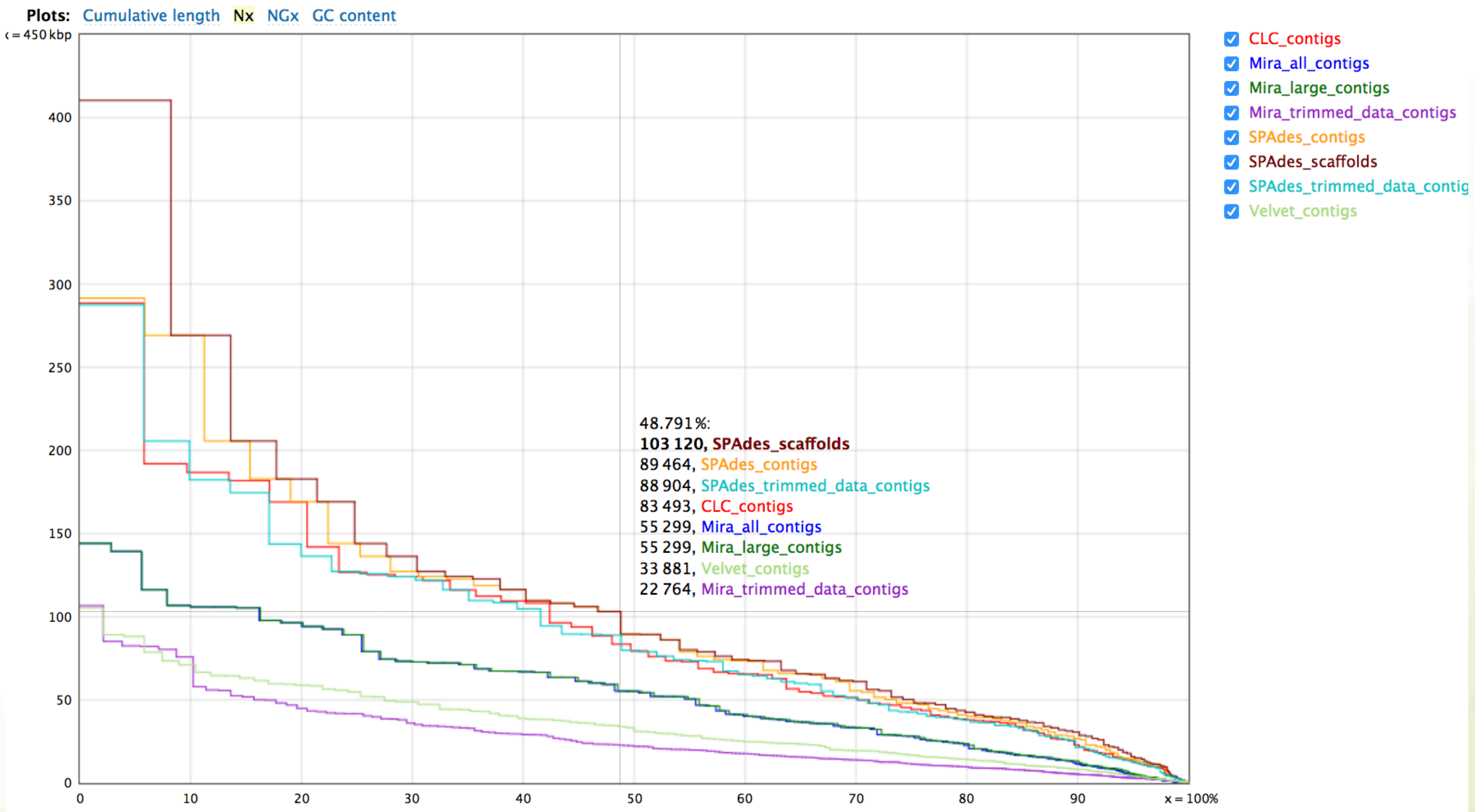
Contigs are ordered from largest (contig #1) to smallest.

Contig graphs



Contigs are ordered from largest (contig #1) to smallest.

Nx graphs



```
48.791%:
103 120, SPAdes_scaffolds
89 464, SPAdes_contigs
88 904, SPAdes_trimmed_data_contigs
83 493, CLC_contigs
55 299, Mira_all_contigs
55 299, Mira_large_contigs
33 881, Velvet_contigs
22 764, Mira_trimmed_data_contigs
```

Now let’s get assembling!