Illumina base calling: Ibis

process image file to extract sample data

create SVM models for each cycle

- train/test data from known genome sequence (e.g. PhiX)
- intensity values of current cycle + previous + next

model outputs

- base call (classification)
- PHRED (like) error probability
  - based on classification probability

Illumina base calling: error probabilities

model–based:

$P_A = I_A/I_A+I_C+I_G+I_T$ (Whiteford et al. 2009)

likelihood of the base call (Das and Vikalo 2012)

(supervised) machine learning:

SVM assignment scores converted to error probabilities using piecewise linear regression (Renaud et al. 2013)

assembly: sequence contigs

an assembly of two or more sequencing reads

usually from different primers or library fragments

[1] confirm the underlying sequence

- disagreement (ambiguous/contradictory bases) must be resolved
  - resolved (as best as possible) based on quality
  - unresolvable coded as IUPAC polymorphism

[2] make consensus (compromise)

- usually larger than individual reads

assembly: super contigs and scaffolds

an assembly of two or more contigs

often produced with a secondary (or tertiary) assembler

- consensus (compromise)
- a major source of error in ‘draft’ genomes

scaffolds

- contain regions of (approximately) known size, but unknown sequence
  - often represented as a uniform size (e.g. 100 Ns)

assembly: quality

N50: median assembly length

- longer is generally better

gene content:

- count the number of reference genes found among the assemblies (BLAST, hmmer, etc.)
- Benchmarking set of Universal Single–Copy Othologs (BUSCO; Seppey et al. 2019)
- Core Eukaryotic Genes (CEG; Parra et al. 2007)

assembly: sequence trimming

sliding window–type algorithm

- [1] read from end
- [2] trim at first window with error below a threshold

window size (usually 20 bp)

allowable ‘error’ threshold

- ambiguous bases (e.g. no more than 2 bases)
- confidence (e.g. no more than 2 bases with QV < 20)

assembly: sequence ‘correction’…

remove ‘systematic’ errors from sequencing reads

rare bits of sequence (kmers) that are similar to common bits of sequence (kmers) are corrected to the more common variant

often improves the sequence quality

- can improve the assembly size
- but decreases the assembly size more often than not

can introduce errors into the sequence

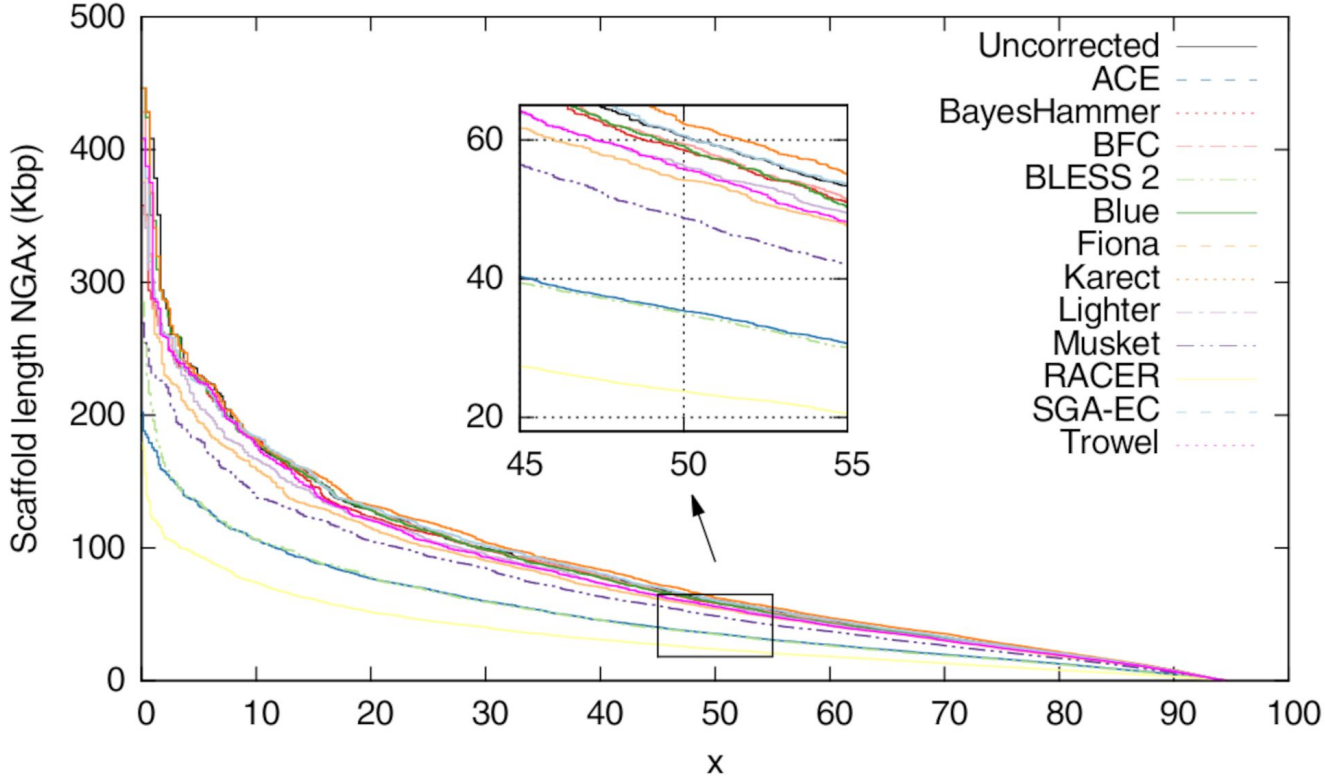


Fig. 2 SPAdes assemblies. SPAdes assembly results for *D. melanogaster* for (un)corrected data. Scaffolds with length NGAx or larger contain x% of the genome

(Heydari et al. 2016; https://doi.org/10.1186/s12859-017-1784-8)

assembly: …sequence ‘correction’

kmer

> e.g. BLESS (Heo et al. 2014), Hammer (Medvedev et al. 2011), HiTEC (Ilie et al. 2011), Musket (Liu et al. 2013), Quake (Qu et al. 2009) RACER (Ilie and Molnar 2013)

multiple sequence alignment (MSA)

> e.g. Coral (Salmela and Schröder 2011), ECHO (Kao et al. 2011), Karect (Allam et al. 2015)

assembly ‘correction’: Quake (kmer)

[1] count (or estimate) kmer frequency

[2] determine common/rare threshold from the data

[3] model common kmers with the Gaussian (or zeta) distribution

[4] model rare kmers with the gamma distribution

[5] for each read remove rare kmers:

- [a] by trimming from the 3' end
- [b] low–quality bases changed to more common bases

assembly ‘correction’: Karect (MSA)

[1] for each read:

- [a] global pairwise align reads to references having at least x kmers (indels are permitted) in common
- [b] store alignment if > y alignment overlap and edit distance < z

[2] for each read:

- [a] extract the shortest stored alignment
- [b] correct to the modal base for each position

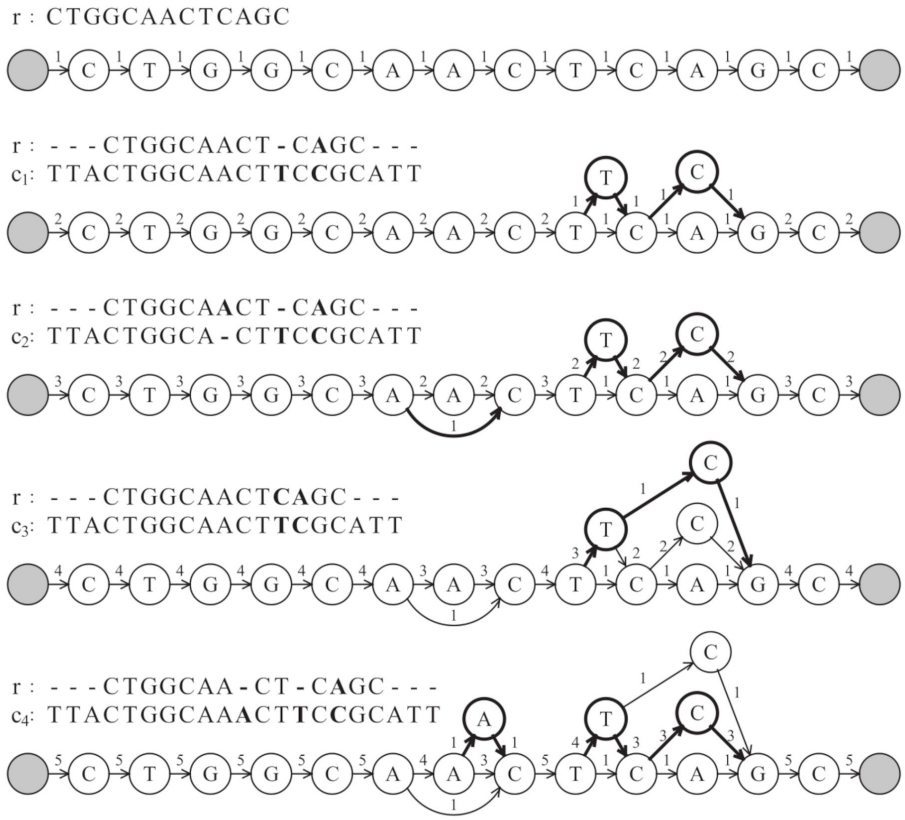


Fig. 2. Example POG. The first row shows the initial POG for reference read *r*. In the second row, c_1 introduces an insertion and a substitution. Next, c_2 includes a deletion, an insertion and a substitution and so on. At each row, the newly introduced changes are shown in bold

(Allam et al. 2015; https://doi.org/10.1093/bioinformatics/btv415)

assembly: types

sequencing by primer walking

- uses PCR products or mapped clones
  - good for known markers, bad for novel genomes
- known location for sequence start => assembly known

shotgun sequencing

- uses a library of overlapping fragments
  - ‘randomly’ sheared or enzymatically cut
- sequence position unknown
  - must locate matching overlapping fragments

assembly: programs…

(too) many programs

program and documentation quality varies greatly

assumed source of data (homogeneous versus heterogeneous)

type of data (read length, single end, paired end)

depth of coverage (== amount of RAM used)

handling of quality values

- if not used in computation, pre–trimming and/or correction is necessary

program choice depends on the ultimate goal

- short accurate versus longer less accurate contings

assembly: …programs…

overlap (local alignment) and consensus

> e.g. Canu (a.k.a. Celera, wgs–assembler; ca. 1999); phrap (Green 1999), Edena (Hernandez et al. 2008), SOAP2/SOAP3 (Li et al. 2009), FALCON (ca. 2014)

prefix tree (partial hash)

> e.g. SSAKE (Warren et al. 2007), VCAKE (Jeck et al. 2007), JR-Assembler (Chu et al. 2013)

de Bruijn graphs

> e.g. Velvet (Zerbino and Birney 2008), SOAPdenovo (Li et al. 2010), Gossamer (Conway et al. 2012), SPAdes (Bankevich et al. 2012), ABySS (Jackman et al. 2017)

assembly: …programs

program choice depends on:

- objective
- data type genome vs. RNAseq vs. ddRADseq etc.
- sequence type(s): read length, paired and/or single end, error level, available quality values, etc.
- amount of data
- RAM efficiency
- quality of resulting output

assembly: overlap and consensus

e.g. phrap (Green 1999)

[0] remove end homopolymers

[1] find reads with matching words (user–defined size)

[2] local alignment of matching reads

[3] exclude vector (defined list) and chimeric reads

[4] compute matching score

[5] place in contigs (start with highest matching score)

[6] make consensus sequence using QV

assembly: prefix trees

e.g. JR-Assembler (Chu et al. 2013)

[0] filter out low complexity (e.g. Dust), low quality

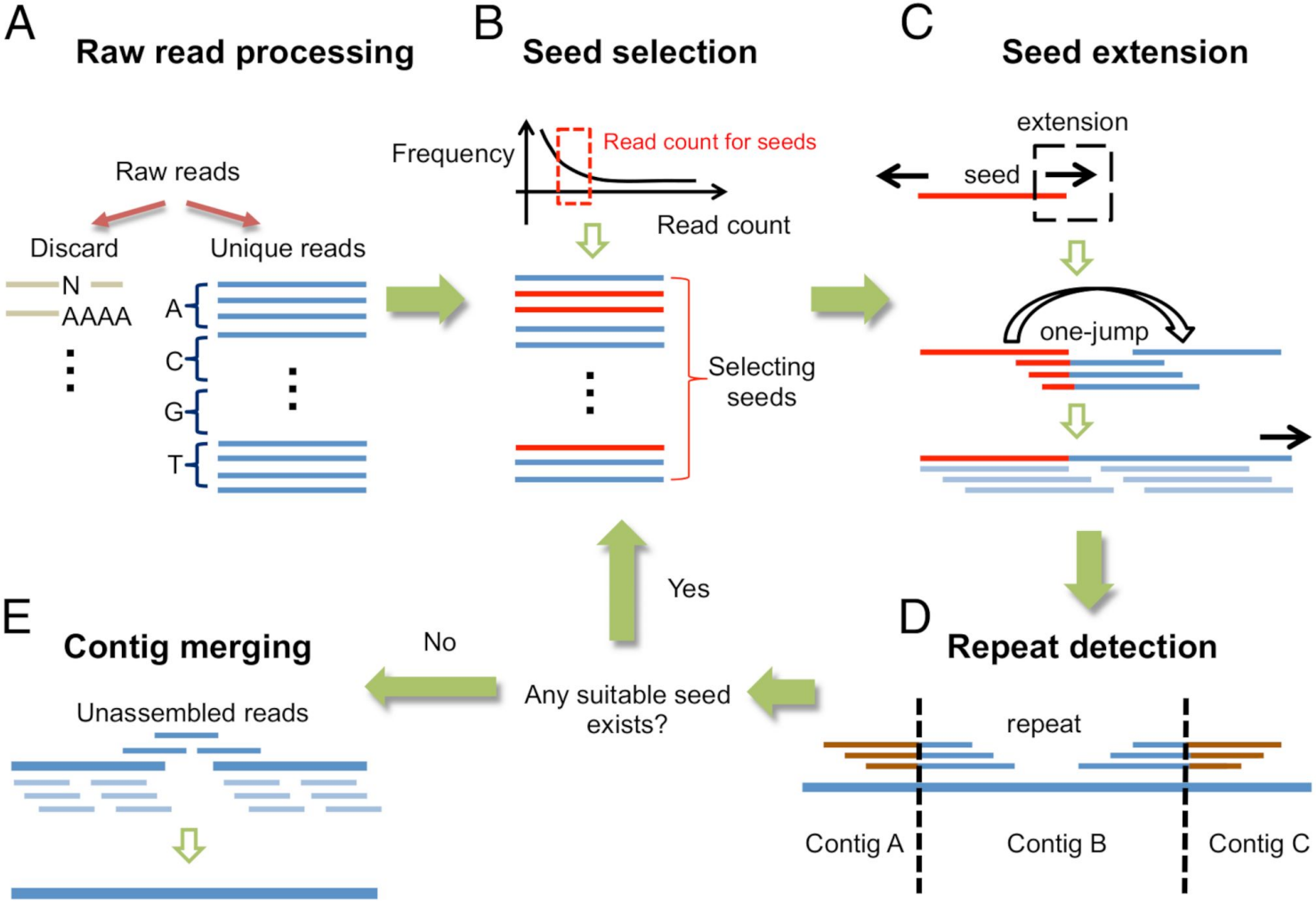
[1] filter out duplicate reads (all orientations; save counts)

[2] select seeds (first quartile of read counts == common)

[3] extend reads with overlapping suffix/prefix (3' then 5')

[4] if extension is impossible, trim 1 base, retry until previous extension is reached

[5] break extension at repeat boundaries



(Chu et al. 2013; https://doi.org/10.1073/pnas.1314090110)

assembly: de Bruijn graphs...

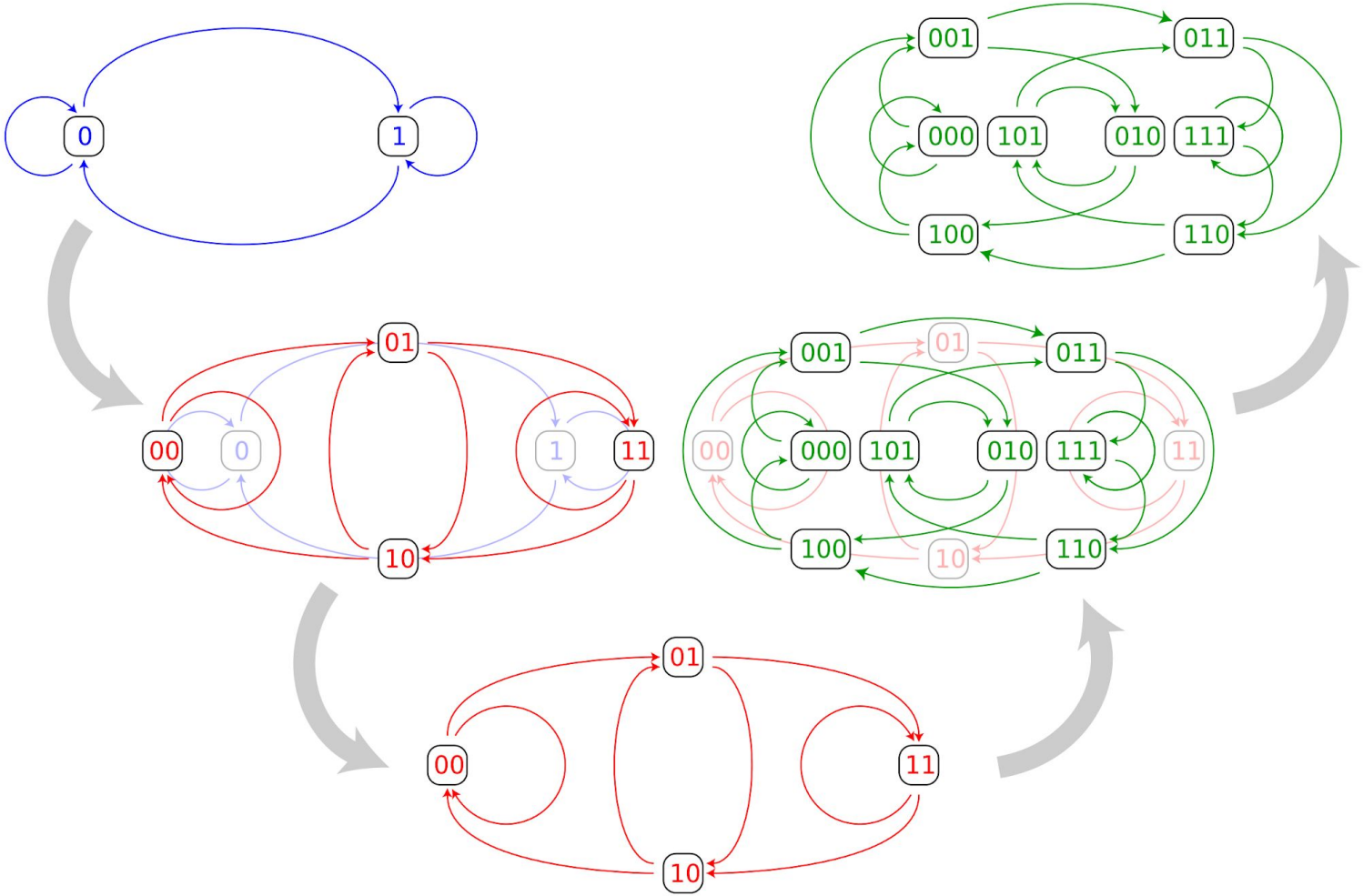
named for Nicolaas Govert de Bruijn (1946)

- discovered independently Irving John Good (1946)
- used by Camille Flye Sainte–Marie (1894)

n–dimensional directed graphs

nodes contain sequence, edges indicate overlap

- an efficient way to store/retrieve DNA sequence reads
- really a precomputed prefix tree



https://commons.wikimedia.org/wiki/File:DeBruijn-as-line-digraph.svg

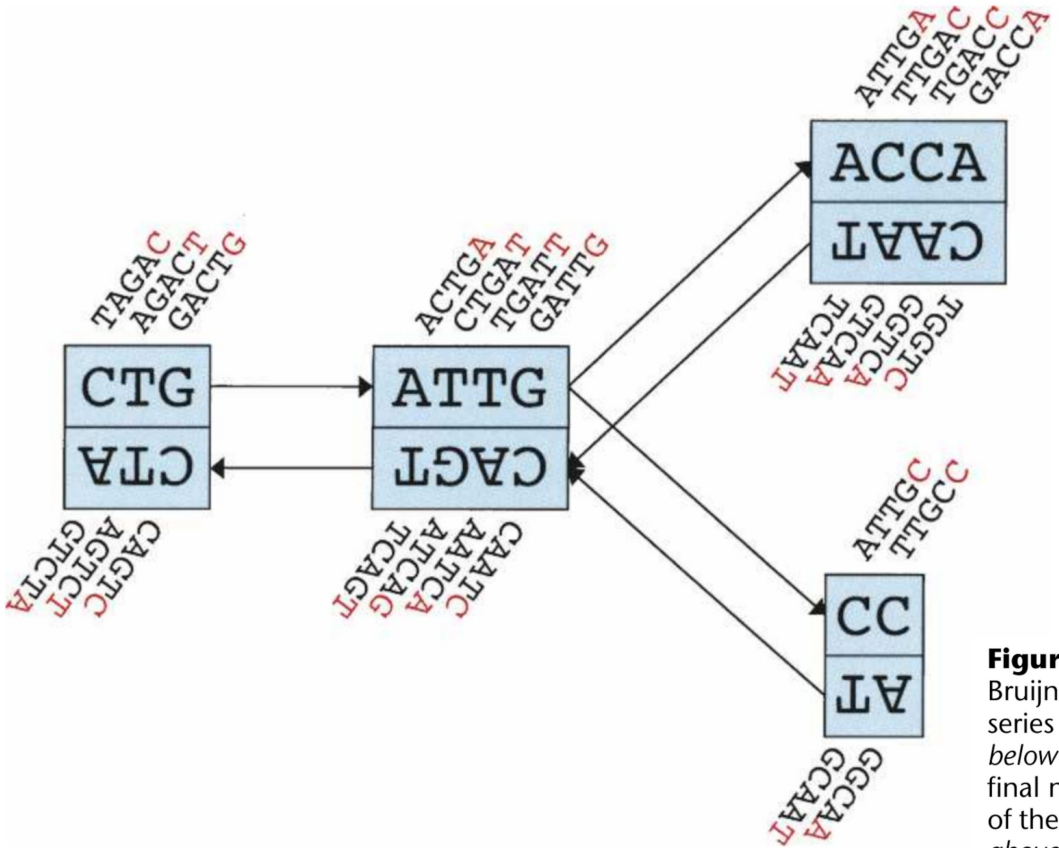


Figure 1. Schematic representation of our implementation of the de Bruijn graph. Each node, represented by a single rectangle, represents a series of overlapping *k*-mers (in this case, $k = 5$), listed directly *above* or *below*. (Red) The last nucleotide of each *k*-mer. The sequence of those final nucleotides, copied in large letters in the rectangle, is the sequence of the node. The twin node, directly attached to the node, either *below* or *above*, represents the reverse series of reverse complement *k*-mers. Arcs are represented as arrows *between* nodes. The last *k*-mer of an arc's origin overlaps with the first of its destination. Each arc has a symmetric arc. Note that the two nodes on the *left* could be merged into one without loss of information, because they form a chain.

(Zerbino and Birney 2008; https://doi.org/10.1101/gr.074492.107)

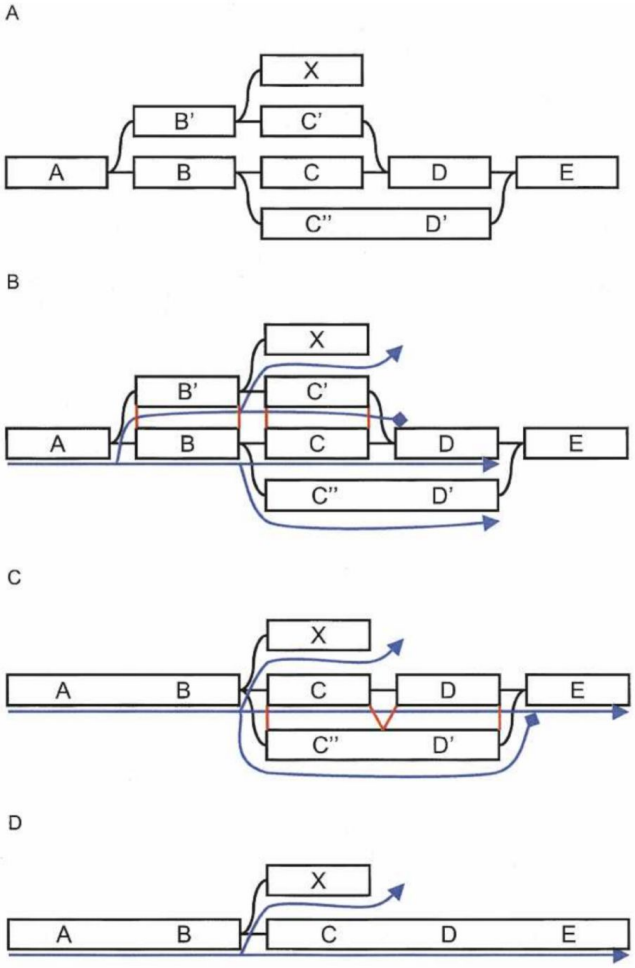


Figure 2. Example of Tour Bus error correction. (*A*) Original graph. (*B*) The search starts from A and spreads toward the *right*. The progression of the *top* path (through B′ and C′) is stopped because D was previously visited. The nucleotide sequences corresponding to the alternate paths B′C′ and BC are extracted from the graph, aligned, and compared. (*C*) The two paths are judged similar, so the longer one, B′C′, is merged into the shorter one, BC. The merging is directed by the alignment of the consensus sequences, indicated in red lines in *B*. Note that node X, which was connected to node B′, is now connected to node B. The search progresses, and the *bottom* path (through C′ and D′) arrives second in E. Once again, the corresponding paths, C′D′ and CD are compared. (*D*) CD and C′D′ are judged similar enough. The longer path is merged into the shorter one.

(Zerbino and Birney 2008; https://doi.org/10.1101/gr.074492.107)

assembly: …de Bruijn graphs...

often ‘compressed’ using alternative representations

- block–sorting compression
  - a.k.a. Burrows–Wheeler transform (1994)
- Succinct de Bruijn graphs (Bowe et al. 2012)
- Bloom filter (Bloom 1970)

assembly: Burrows–Wheeler…

^ACGTACGT$		^ACGTACGT$		$		
$^ACGTACGT		$^ACGTACGT		T		
T$^ACGTACG		ACGT$^ACGT		T		
GT$^ACGTAC		ACGTACGT$^		^		
CGT$^ACGTA		CGT$^ACGTA		A		
ACGT$^ACGT	-sort->	CGTACGT$^A	-extract->	A	-encode->	T_2^A_2C_2G_2$
TACGT$^ACG		GT$^ACGTAC		C		
GTACGT$^AC		GTACGT$^AC		C		
CGTACGT$^A		T$^ACGTACG		G		
ACGTACGT$^		TACGT$^ACG		G		

assembly: …Burrows–Wheeler

```
$          $^          $^A          $^AC                ^ACGTACGT$
T          T$          T$^          T$^A                $^ACGTACGT
T          TA          TAC          TACG                ACGT$^ACGT
^          ^A          ^AC          ^ACG                ACGTACGT$^
A          AC          ACG          ACGT                CGT$^ACGTA
A  - += -> AC  - += -> ACG  - += -> ACGT  ->>> -sort-> CGTACGT$^A
C   sort   CG   sort   CGT   sort   CGT$                GT$^ACGTAC
C          CG          CGT          CGTA                GTACGT$^AC
G          GT          GT$          GT$^                T$^ACGTACG
G          GT          GTA          GTAC                TACGT$^ACG
```

assembly: Bloom (1970) filter

an imprecise hash method of storing data

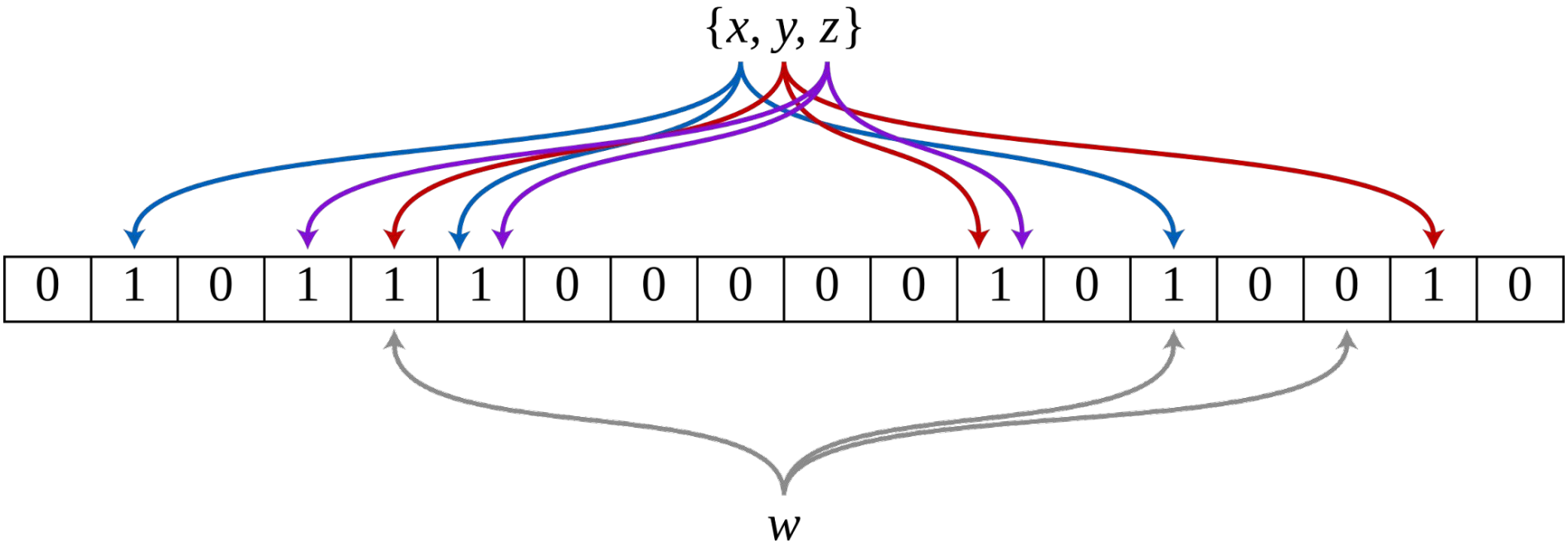
- no false negatives, but false positives occur
- useful when most queries will be negative
  - i.e. most reads do not have most kmers

filter size a function of kmers (4^k) and acceptable error rate

input is processed with *n* hash functions

- [emulated with one high–dispersion (and long output) function]

hash values determine where (and which) bits to store



https://commons.wikimedia.org/wiki/File:Bloom_filter.svg

assembly: …de Bruijn graphs…

e.g. ABySS 1.0 (Simpson et al. 2009), a short read assembler

[0] quality trim sequences, remove ambiguous reads

[1] make graph of kmers

(kmer size usually 24-32, must be shorter than read length)

[2] create hash table of overlaps (i.e. the de Bruijn graph)

[3] simplify the graph

[a] delete dead ends

[b] use near matches if needed

[4] extract sequence by traversing the graph

assembly: de Bruijn graphs…

e.g. Flye (Kolmogorov et al. 2019), a long read assembler

[0] construct pairwise local alignments of all reads

[1] make graph of aligned segments (i.e. the de Bruijn graph)

[2] extend segments based on an overlap threshold

[3] simplify the graph

- [a] delete dead ends
- [b] use read frequency to choose ‘correct’ path

[4] extract sequence by traversing the graph

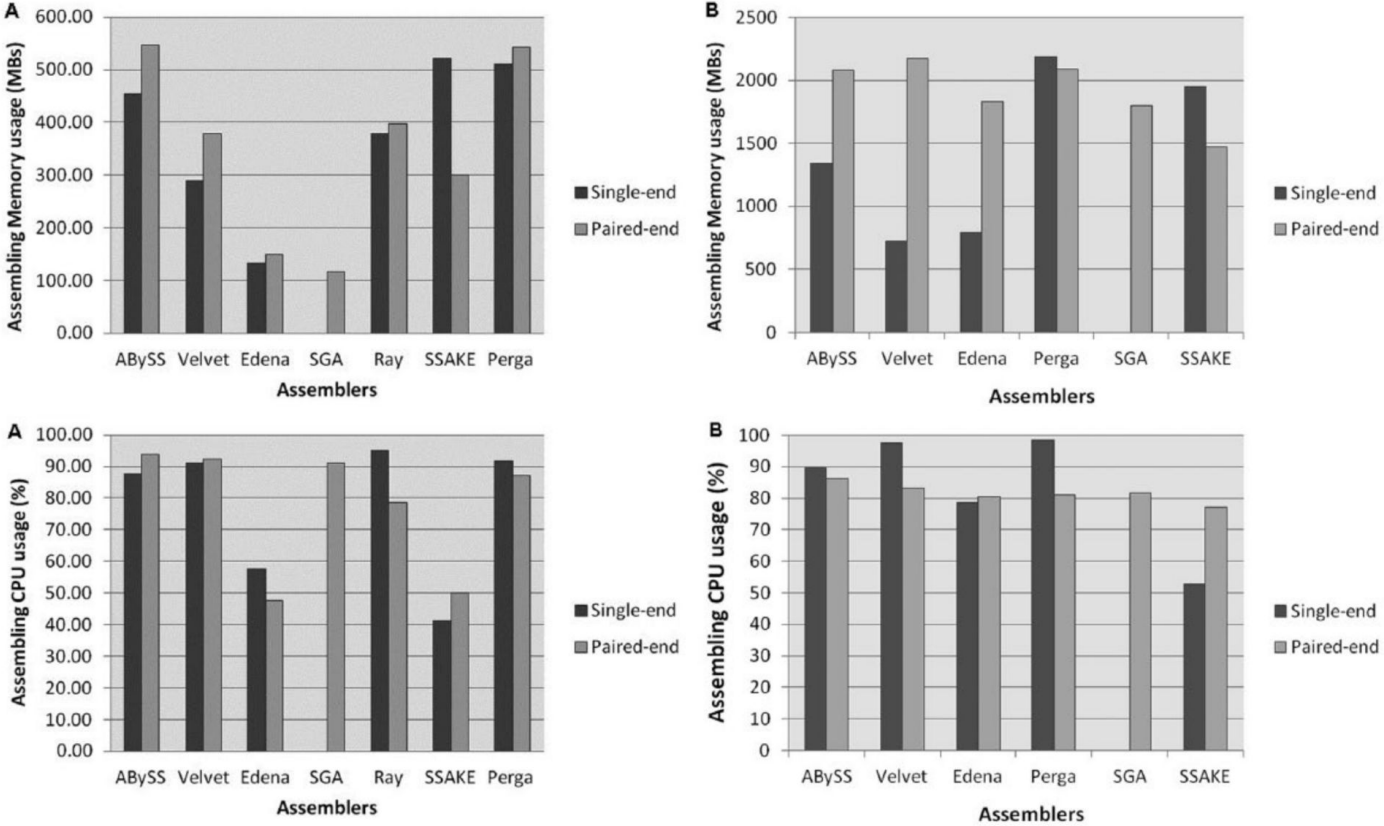


Figure 2. The mean comparison of memory usage and CPU usage of each assembler for (A) paired-end and single-end prokaryotic data sets and (B) paired-end and single-end eukaryotic data sets.

(Khan et al. 2018; https://doi.org/10.1177/1176934318758650)

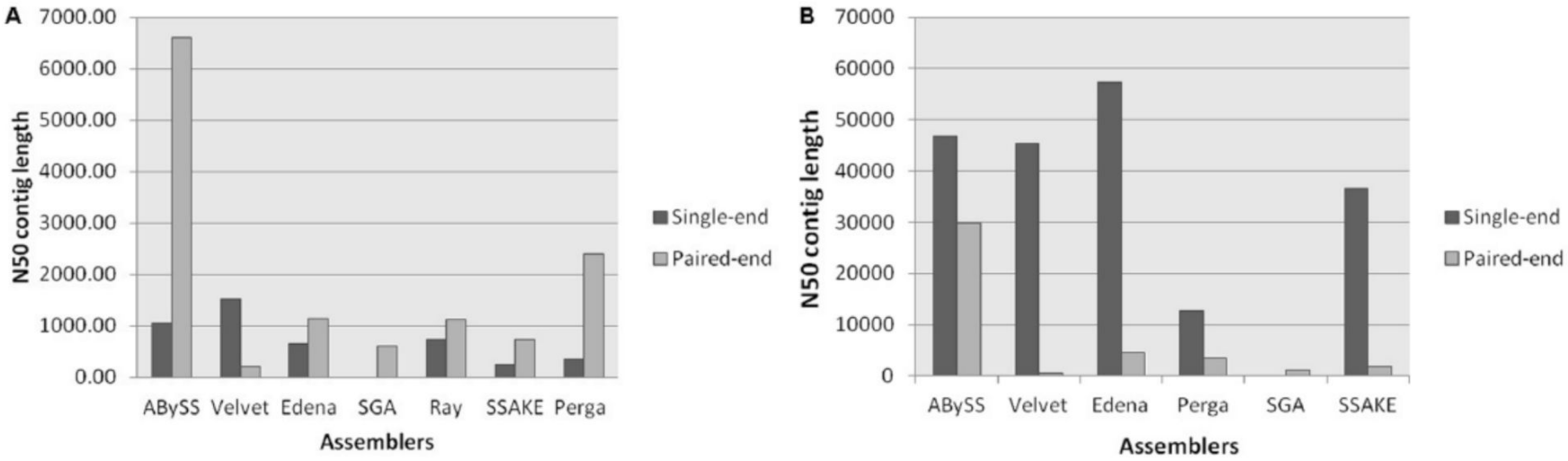


Figure 4. The comparison of the N50 contig length by median of each assembler for (A) paired-end and single-end prokaryotic data sets and (B) paired-end and single-end eukaryotic data sets.

(Khan et al. 2018; https://doi.org/10.1177/1176934318758650)

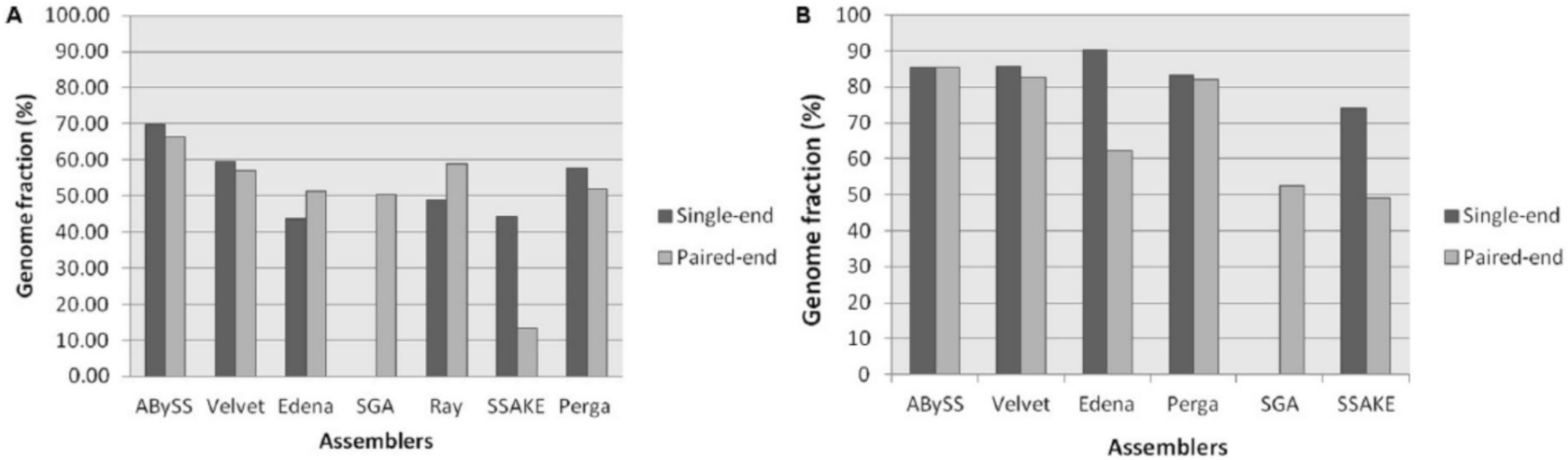


Figure 5. The comparison of mean genome fraction of each assembler for (A) paired-end and single-end prokaryotic data sets and (B) paired-end and single-end eukaryotic data sets.

(Khan et al. 2018; https://doi.org/10.1177/1176934318758650)

assembly: trinity (Grabherr et al. 2011)

the ‘best’ (most commonly used) transcriptome assembler

genome–guided *de novo* assembly

a perl script wrapper around many other programs

    jellyfish, inchworm, chrysalis, butterfly, TransDecoder, etc.

[0; inchworm] prefix tree assembly

[1; chrysalis] de Bruijn graph of [0]

    capture isoforms and paralogs

[2; butterfly] simplifies and extracts from [1]

    filters out ‘implausible’ isoforms

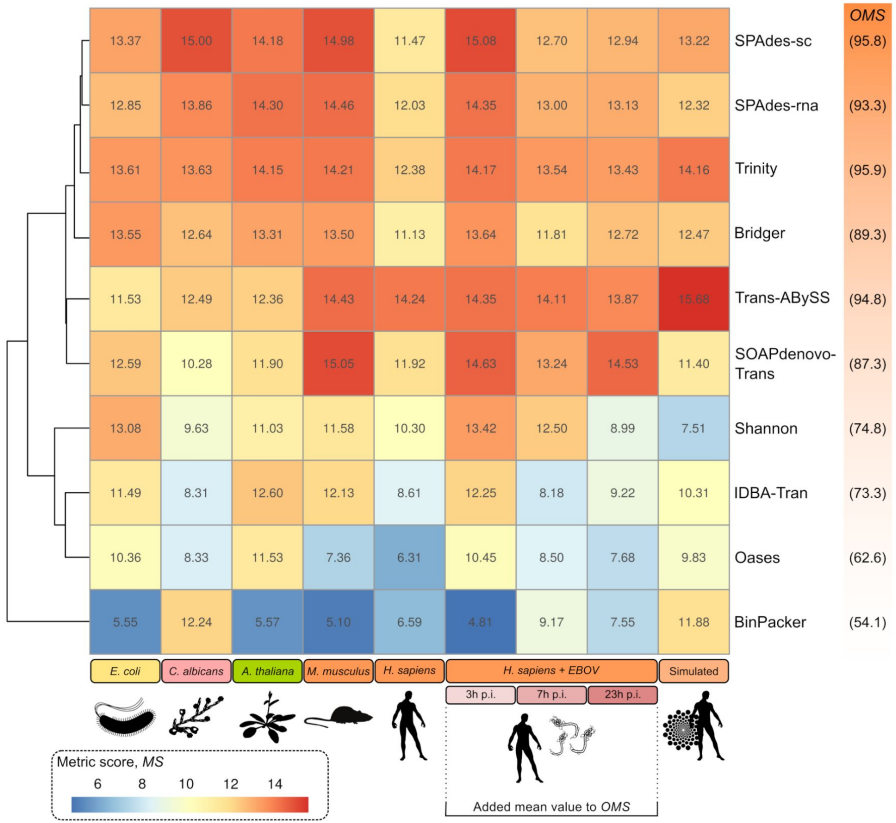


Figure 2: Heat map showing for each data set (column) and each assembler (row) the calculated metric score (MS) (detailed definition in the Methods). The assembly tools are clustered based on their achieved MS over all data sets. The MS for 1 assembly tool and a single data set is based on 20 pre-selected metrics (see Table 4 and Methods for details) and is shown in 1 cell in the heat map (e.g., the MS for *E. coli* and `Trinity` [10] is 13.61). For each data set, an assembler's MS is the sum of (0,1)-normalized scores of every single metric. The hierarchical clustering of the metric scores divides the assembly tools into 2 groups of generally high-ranked (upper half) and low-ranked (bottom half) tools. Except for `Trans-ABySS` [9], the MS reached for the largest human RNA-Seq data set is generally lower. Numbers in brackets next to the assembler names present the summarized metric scores (overall metric score, OMS) for all 9 data sets (see Methods). For the 3 similar human data sets infected with EBOV (Fig. 1), we added the mean MS value to the OMS. Details about the metric results for the human data set (no infection) can be found in Table 2 and for all other data sets Electronic Supplement Table S10.

(Holzer and Marz 2019; https://doi.org/10.1177/1176934318758650)

assembly: How good is it?

compare among assemblers

number of ‘genes’, number of contigs, N_{50}

compare to a reference (if possible)

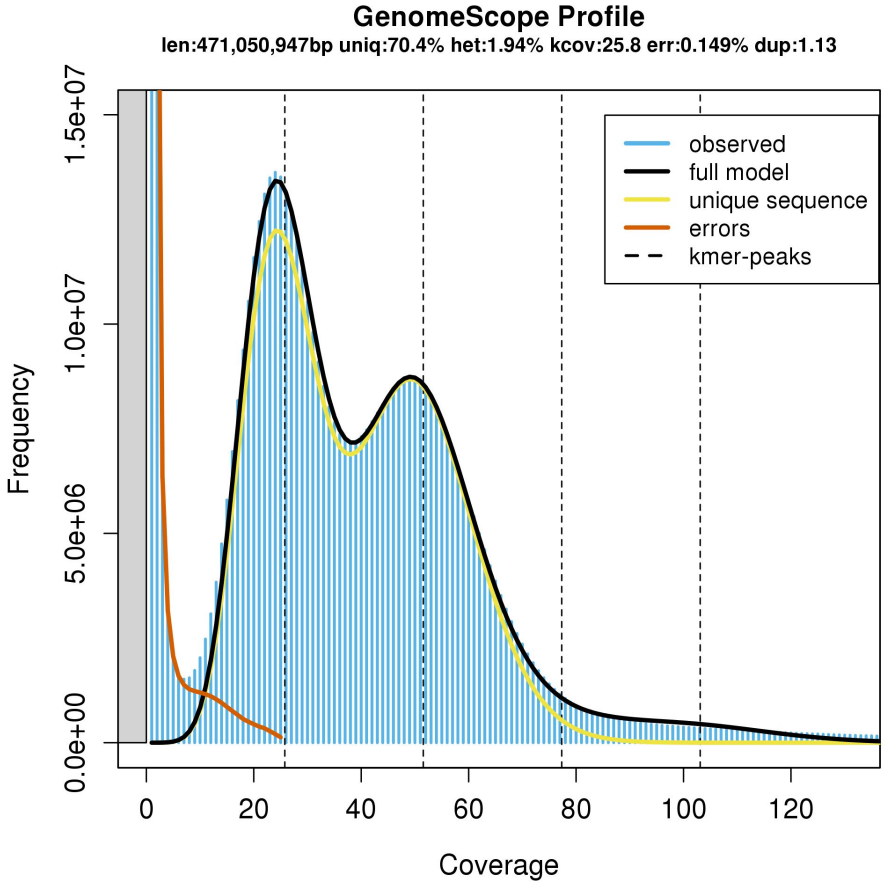
compare to expected gene content (e.g. BUSCO)

compare to expected genome size

assembly: How good is it?

GenomeScope (Vurture et al. 2017):

- genome size, heterozygosity, and repeat content
- kmer estimates (21–mers; counted via jellyfish)
  - >25x coverage required
  - <2% low error rate required (e.g. Illumina)
- statistical model using the kmer histogram:
  - excludes low frequency kmers
  - homozygous (Poisson) vs. heterozygous (bimodal)
    - repeats are outside the ideal distribution
  - genome size = normalized observed vs. mean coverage



(Vurture et al. 2017; https://doi.org/10.1093/bioinformatics/btx153)

read mapping

placing reads onto reference sequences

- to quantify expression
- to find minor variants (re–sequencing)
- to identify taxa (metabarcoding)

assumes that reference sequences are (mostly) correct

- a local alignment (with or without QV consideration)
  - if QV is not used, pre–trimming and/or correction should be used

many, many, programs

speed and accuracy are highly variable

tuning is often required

read mapping: STAR (Dobin et al. 2013)

[0] for each read

search for maximal mappable prefix

if match is interrupted (a ‘splice site’) => split read

[1] use contiguous mapped reads as local alignment seeds

[2] select unmapped reads using a prefix tree

[3] join unmapped and mapped using local alignment

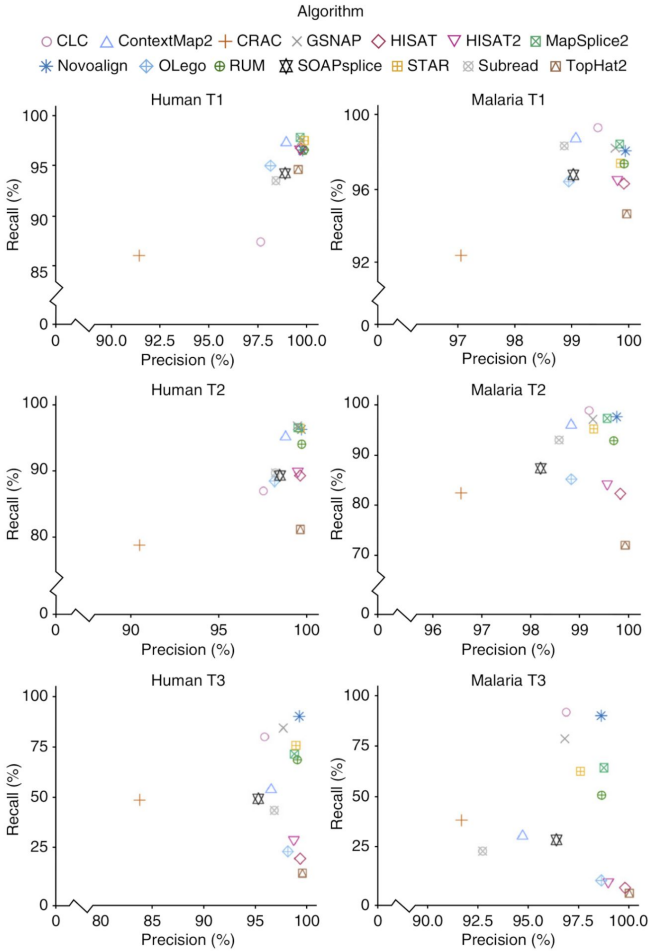
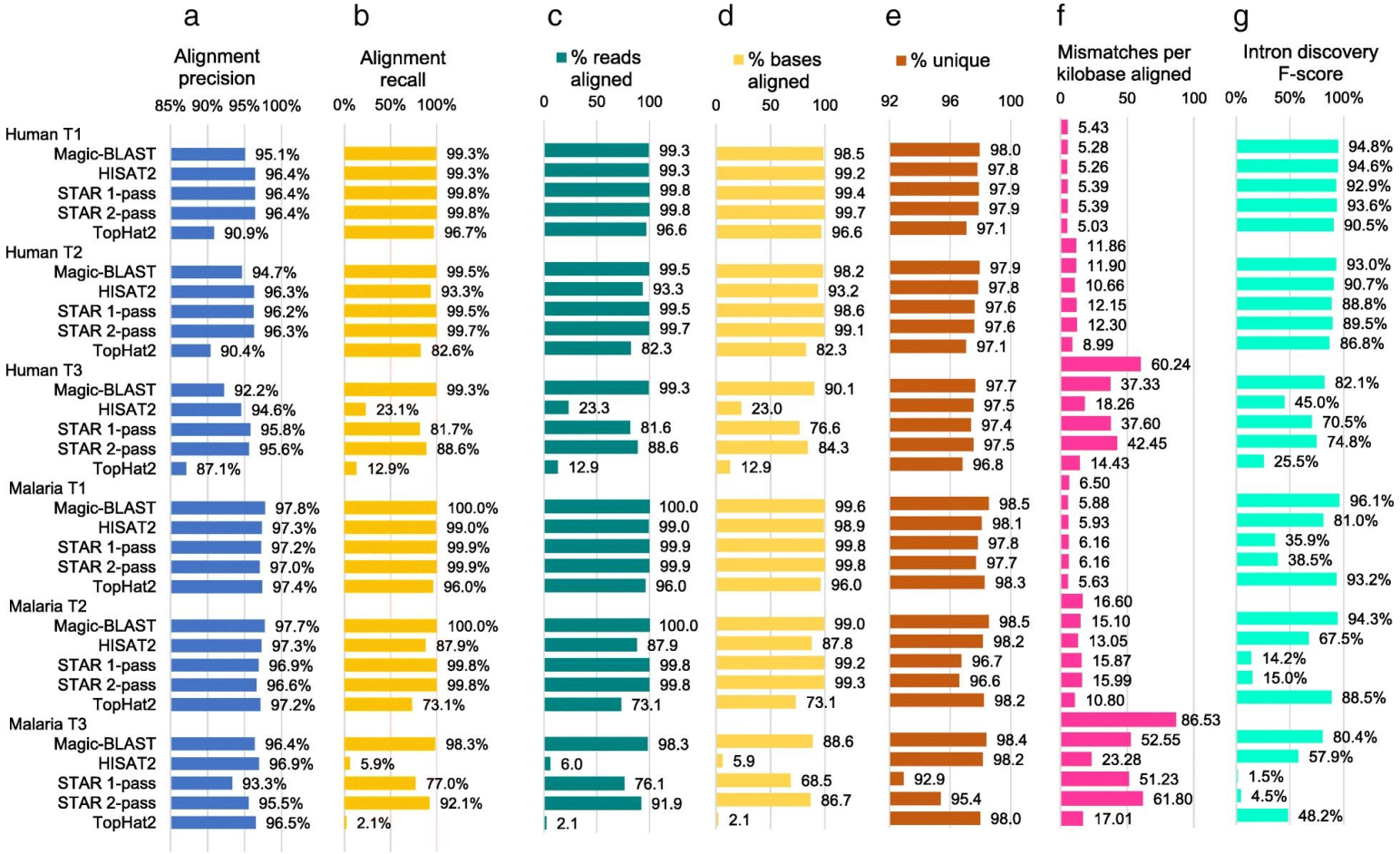


Figure 1 | Base-level precision and recall for human and malaria data sets.

(Baruzzo et al. 2016; https://doi.org/0.1038/nmeth.4106)



(Boratyn et al. 2019; https://doi.org/10.1186/s12859-019-2996-x)