Contiguity: Contig adjacency graph construction and visualisation.

Mitchell J. Sullivan[a], Nouri L. Ben Zakour[a], Brian M. Forde[a], Mitchell Stanton-Cook[a], Scott A. Beatson[a]*

[a]Australian Infectious Diseases Research Centre, and School of Chemistry and Molecular Biosciences, The University of Queensland, Brisbane, QLD 4072, Australia

Running Head: Contiguity: Assembly visualisation

*To whom correspondence should be addressed. Tel: +61 7 3365 4863; Email: s.beatson@uq.edu.au

Keywords: Chromosome Mapping, Computational Biology, Computer Graphics, Genetics, Comparative genomics, Computational Biology, Gap closing, Contig ordering, Genome analysis, Bioinformatics

Abstract

Contiguity is interactive software for the visualization and manipulation of *de novo* genome assemblies. Contiguity creates and displays information on contig adjacency which is contextualized by the simultaneous display of a comparison between assembled contigs and reference sequence. Where scaffolders allow unambiguous connections between contigs to be resolved into a single scaffold, Contiguity allows the user to create all potential scaffolds in ambiguous regions of the genome. This enables the resolution of novel sequence or structural variants from the assembly. In addition, Contiguity provides a sequencing and assembly agnostic approach for the creation of contig adjacency graphs. To maximize the number of contig adjacencies determined, Contiguity combines information from read pair mappings, sequence overlap and De Bruijn graph exploration. We demonstrate how highly sensitive graphs can be achieved using this method. Contig adjacency graphs allow the user to visualize potential arrangements of contigs in unresolvable areas of the genome. By combining adjacency information with comparative genomics, Contiguity provides an intuitive approach for exploring and improving sequence assemblies. It is also useful in guiding manual closure of long read sequence assemblies. Contiguity is an open source application, implemented using Python and the Tkinter GUI package that can run on any Unix, OSX and Windows operating system. It has been designed and optimized for bacterial assemblies. Contiguity is available at http://mjsull.github.io/Contiguity .

Introduction

The emergence of high-throughput sequencing technologies has led to a massive increase in the number of unassembled or draft bacterial genome sequence data sets [1]. *De novo* assembly of sequencing reads produced using high-throughput sequencing methods often results in highly fragmented assemblies containing hundreds of contiguous sequences (contigs). Although long reads, such as those produced by Pacific Bioscience's single molecule real time sequencing (SMRT), significantly reduce fragmentation in bacterial genome assemblies, they frequently do not assemble into a single contig [2]. Consequently, contig ordering, scaffolding, identification of spurious or misassembled contigs and comparative analysis of an assembly all remain time-limiting steps during the analysis of a *de novo* assembly.

Several tools exist that allow easy visualization of pairwise or multiple alignments, including Easyfig [3], Artemis Comparison Tool [4], genoPlotR [5], Interactive Genomics Viewer [6] and Mauve [7]. These tools allow the rapid identification of structural variations between two sequences such as rearrangements, insertions, and deletions. Many of these events may be biologically important and can be a result of prophages, plasmids and other mobile genetic elements. Such events account for much of the variation in bacterial species such as *Escherichia coli* [8]. However, mobile genetic elements are relatively difficult to resolve in draft or metagenome assemblies primarily due to an abundance of insertion sequences within these elements that result in collapsed repeats and a lack of specific information about contig adjacency. Mobile genetic elements often assemble into several contigs making it unclear whether several contigs with novel sequence are part of the same mobile genetic element, or belong to several distinct elements.

In theory, mobile genetic elements and other difficult to assemble genomic regions can be reconstructed by examining contig interconnectivity within an assembly. By determining which contigs are adjacent to one another in the underlying assembly graph, potential arrangements of those contigs in context of the complete genome can be determined. This allows the use of synteny to contextualise sequence that is not present in complete reference genomes and can also help determine the sequence of genomic regions that span multiple contigs. Adjacency information can also be used to group contigs into distinct elements, such as chromosomal and extra-chromosomal DNA. This approach is used by PLACNET [9] to identify plasmid contigs in *de novo* assembled genomes. PLACNET creates an undirected graph of contig adjacencies that can be visualized with a tool such as Cytoscape [10]. Using such an approach, specific information about order and orientation of contigs in the plasmid, relative to one another, cannot be inferred. Several methods exist for finding interconnectivity between contigs, such as looking at paired-end reads shared by contigs in a *de novo* assembly or using transcript data. This information can be leveraged by scaffolding algorithms, such as SOPRA [11] and SSPACE [12], to improve *de novo* assemblies by joining connected contigs where no ambiguity exists. However, scaffolding can introduce errors into assemblies and provides no information about potential adjacencies between contigs in regions that are unable to be resolved, such as repetitive regions of the chromosome. Interconnectivity can also be visualized using programs such as Consed [13], Phrapview [14], Abyss-explorer [15], TGnet [16], ContigScape [17] and Bandage [18]. Consed and Phrapview display a linear relationship between contigs with connections between contigs being inferred from paired reads. Abyss-explorer, TGnet and ContigScape display assemblies as a directed graph. Abyss-explorer infers connectivity from graph information and read pair information provided by the De Bruijn assembler Abyss [19]. TGnet finds adjacencies using transcript information, and Contigscape infers adjacencies by identifying reads shared between contigs assembled by the "Newbler Assembler" or connectivity using paired reads. Bandage can be used to visualize the LastGraph file produced by the Velvet assembler [20], FASTG files and Trinity.fasta files produced by the RNA-seq assembler Trinity [21].

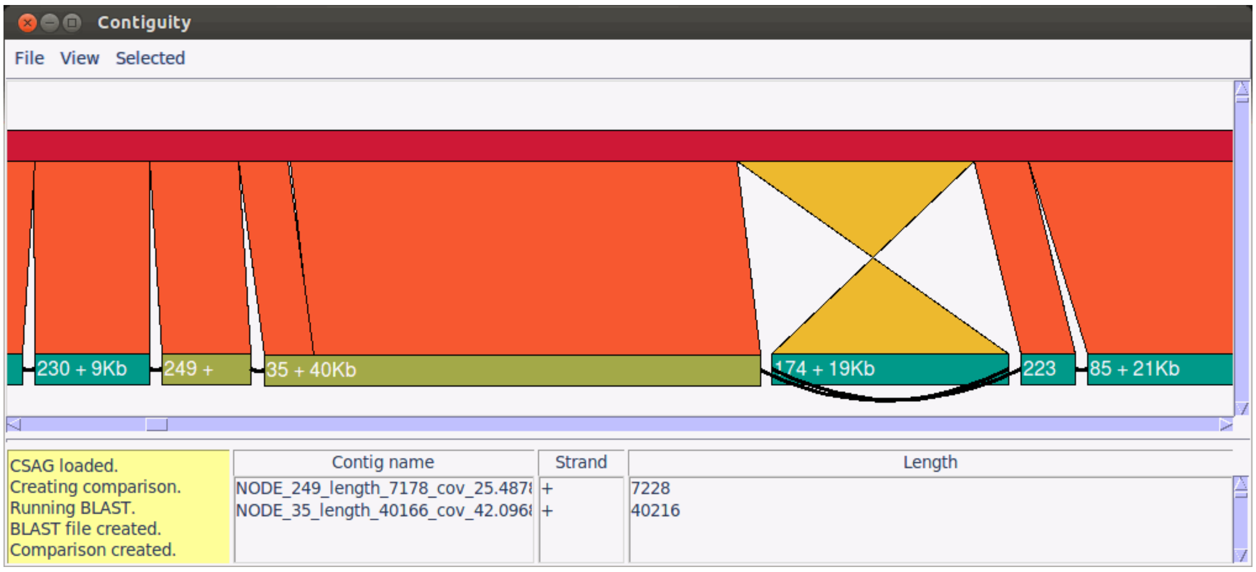
These methods, described above, are limited to creating graphs from specific data types that are not always available to the end user. Alternatively, they require the use of a specific assembly program, which may result in a suboptimal assembly. Graphs based on the output of an assembler also prevent the user from performing additional optimization of their assemblies, such as scaffolding or misassembly correction. Assemblies often result in hundreds of contigs, with each contig typically having between 2 to 4 connections to other contigs. Although small assemblies can be displayed concisely, as assembly size grows visual representations of the graph can quickly become cluttered making it difficult to extract meaningful information.

Contiguity makes contig adjacency graphs more accessible to users unfamiliar with the concept. Contiguity adds sequence comparison to the visualization of contig adjacency graphs. This allows the user to contextualize contig adjacency information with similarity and the order inferred from a reference sequence. The user can quickly and easily identify genome rearrangements, insertions, deletions and potential misassemblies. Contiguity includes a purpose built contig adjacency graph creation algorithm that combines existing approaches and allows adjacency graphs to be built from any assembly irrespective of sequencing or assembly method. In addition to a description of Contiguity's core functionality (the construction and representation of contig adjacency graphs) we also provide two case studies of existing projects in which Contiguity has been used to improve assembly and elucidate structural rearrangements. Contiguity is an open source project implemented in Python using the Tkinter graphical user interface library, it available on Windows, OSX and GNU/Linux.

Methods and results

Contiguity overview

Contiguity is designed to enable the visualization and organization of *de novo* assemblies. It allows both comparison information and contig adjacency graph information to be visualized simultaneously using the same BLAST comparison format used in tools such as Artemis Comparison Tool [4] and Easyfig [3] (Figure 1).



```
CSAG loaded.
Creating comparison.
Running BLAST.
BLAST file created.
Comparison created.
```

Contig name	Strand	Length
NODE_249_length_7178_cov_25.487	+	7228
NODE_35_length_40166_cov_42.096	+	40216

Figure 1: Contiguity main window. Visualization and organization is achieved in the central canvas of the main window, contigs are shown as teal (gold when selected) rectangles with their name, orientation and length displayed within the box (if enough room exists). Sequence alignments to a reference are shown as orange (yellow for inverted) polygons, reference sequences are shown as red rectangles. Contig adjacency is shown using black arcs. The yellow console in the bottom left corner shows information on currently running processes. The grey panels in the bottom right show information on currently selected contigs.

Contiguity uses the BLAST comparison results to order contigs according to a reference and a graph file (such as LastGraph from Velvet) to show connections between adjacent contigs. The visual and interactive layout of contigs allows the user to easily order and scaffold contigs into a putative chromosome, whilst identifying potential regions of structural variation. Optionally, Contiguity can create a custom contig adjacency graph (Contiguity-CAG) using read files and an assembled contig/scaffold file. The Contiguity-CAG improves on the sensitivity of current methods by utilizing three complementary approaches (paired-read, overlap and De Bruijn graph exploration). Furthermore, the Contiguity-CAG representation is both sequencing type and assembly method independent and enables the sequence between connected contigs to be determined. It merges information from De Bruijn graph exploration, overlap searches and paired-end mapping to create a highly-sensitive graph of contig adjacencies.

Contiguity Graphical User Interface Layout

Sequence assemblies can be loaded into the Contiguity Graphical User Interface (GUI) from either FASTA, LastGraph, ACE, DOT, FASTG or a Contiguity-CAG file (Figure 1). Contig adjacency information stored within LastGraph, ACE, FASTG, DOT and Contiguity-CAG files is also loaded. Contiguity-CAG files can be created from the drop-down menu. A comparison to a reference can be generated from within Contiguity using NCBI-BLAST+ [22], if BLAST+ binaries are found in the users path. Alternatively, the BLAST comparison can be loaded from a file, provided that it is in the standard BLAST tab format. Contigs and their adjacencies can then be viewed on a canvas in the main window; the user can choose to display all contigs, only contigs with BLAST matches to the reference sequence, only contigs with no hits to the reference or a user-defined subset of contigs. Contigs are ordered firstly by hits to the reference, then by connectivity information, placing adjacent contigs next to each other where possible, and finally contigs with no hits or adjacency information are placed on the canvas from longest to shortest. Once contigs are loaded, sequence alignments between and/or within contigs can be created and visualized to highlight repetitive or duplicated regions as blue (or orange for inverted) ribbons.

Contiguity's canvas can be zoomed, stretched and shrunk in the X-dimension. Contigs and reference sequences can be independently moved anywhere on the canvas and duplicated, removed or reversed interactively. Paths between contigs connected in the adjacency graph can be found manually or by an implementation of a depth-limited search according to a user-defined cut-off.

The user can colour contigs by using metadata such as coverage, GC content or AT and GC skew. User-created lists can be used to order and colour contigs. The subgraph can be easily expanded or rearranged using context-driven menus. Specific edges or alignments can also be highlighted from these menus.

Contigs in the Contiguity-CAG can be selected automatically or interactively, and written as a multi-FASTA and/or as a single scaffold. Videos demonstrating the functionality of Contiguity and a comprehensive manual are available at http://mjsull.github.io/Contiguity/.

Contiguity Contig Adjacency Graph (Contiguity-CAG) creation

A CAG is a directed graph where each contig in a *de novo* assembly is represented as two nodes (forward and reverse strand). A directed edge is created from node A to node B if the sequence represented by node B occurs directly after the sequence represented by node A.

How a genome will be represented as a *de novo* assembly is due to a combination of sequencing type, composition (location and number of repeat regions) and the assembler used (Figure 2).

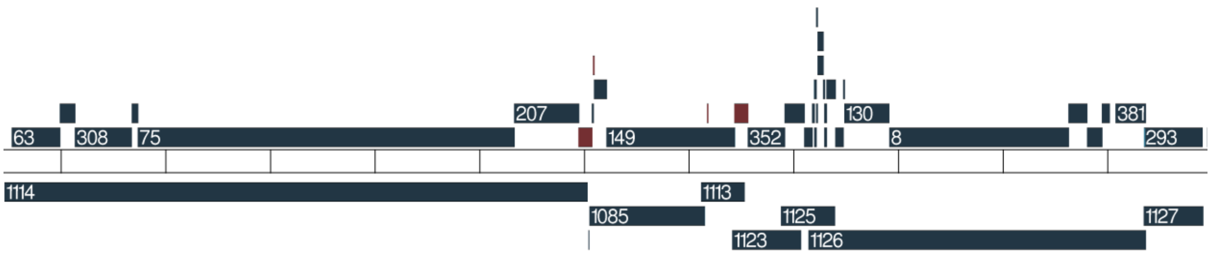


Figure 2: Anatomy of an assembly. An Illumina sequencing run of *Escherichia coli* str. UTI89 was assembled with Velvet and Abyss. Contigs were then mapped to the 110 Kbp plasmid found in UTI89. Velvet (top) produces highly fragmented repetitive regions. Abyss (bottom) assembles repetitive regions into long contiguous segments, however this results in large overlaps between contigs. Contig names are displayed where they fit within the contig borders.

Adjacent contigs can overlap by hundreds of bases to a single base, be directly adjacent to one another or have unassembled sequence between them. Contiguity attempts to reconstruct an adjacency graph by i) finding overlapping contig edges and ii) performing an end-to-end De Bruijn search – a method that uses the proximity of contig ends in the De Bruijn graph to infer contig adjacency. Contigs adjacencies, where the overlap is greater than the *k*-mer value, are found by identifying overlapping edges of the contig. Contigs that overlap by less than or equal to the chosen *k*-mer value or do not overlap are found by searching through a De Bruijn graph. This approach allows Contiguity to predict sequence or overlap size between adjacent contigs. When scaffolds are reconstructed, using the GUI, this information can optionally be used to reconstruct scaffolds that more accurately represent the original genome. This feature is provided with the disclaimer that all predicted joins between contigs are "best guesses" and should be independently verified with PCR or by mapping reads onto the reconstructed scaffolds. Unfortunately this approach has difficulties when encountering sequence specific error profiles in Illumina sequencing. Sequence specific error profiles occur when sequences dephase due to sequence-specific interference of the base elongation process during sequencing [23]. This causes regions with extremely low quality bases. If filtering or trimming is applied before assembly, this will result in regions with no or low coverage. Although these regions cannot be traversed by a De Bruijn graph, they are often bridged by paired-end reads. iii) Paired read information is also used to identify adjacent contigs, to further improve the sensitivity of Contiguity's approach. This approach cannot find sequence between adjacent contigs or accurately determine the size of an overlap. In the final step of the adjacency graph creation, the edges found by all three methods are merged, exploiting the advantage of the De Bruijn/overlap approach and a paired approach. (Figure 3).

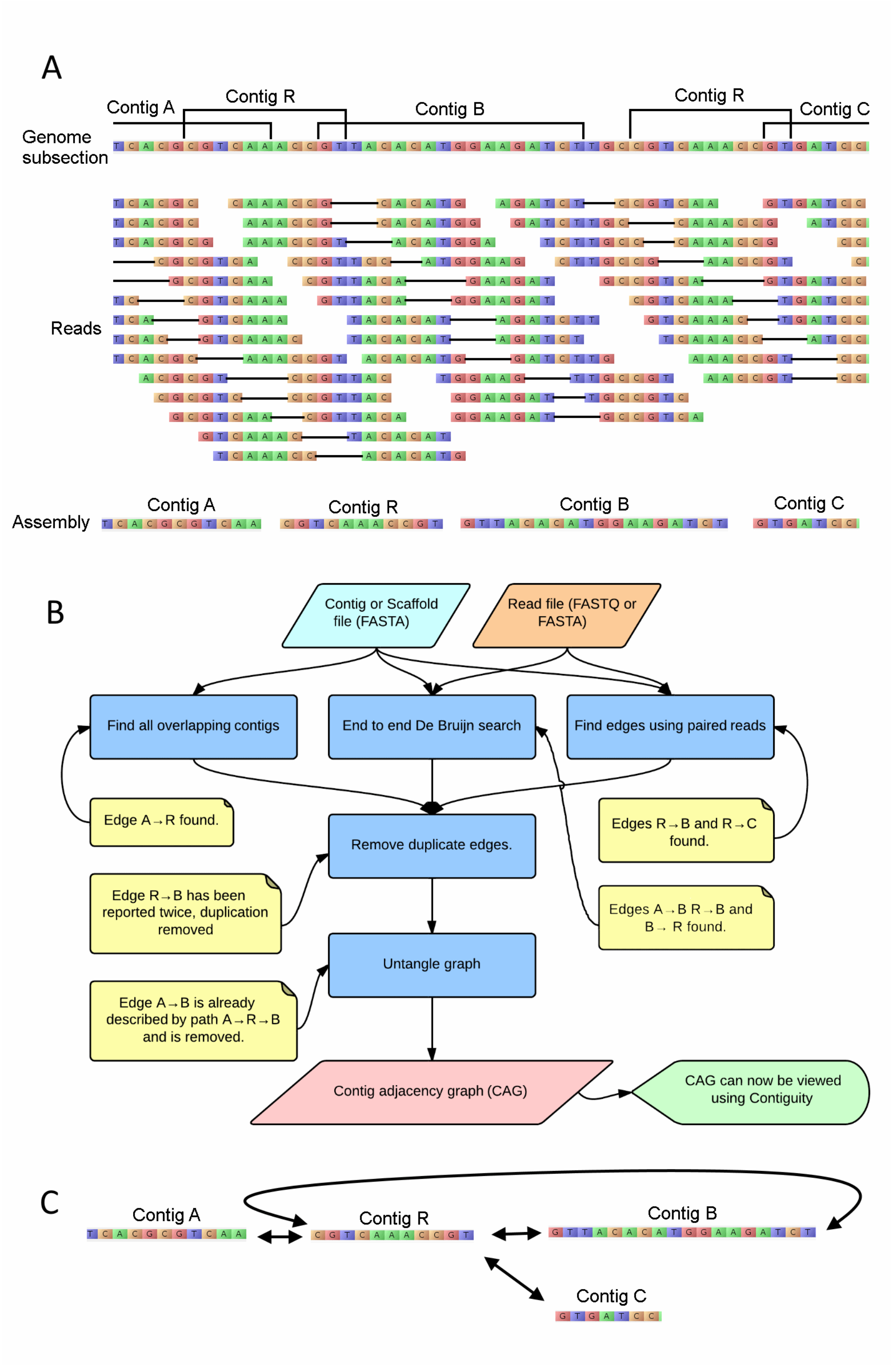

A
Contig A
Contig R
Contig B
Contig R
Contig C
Genome subsection
Reads
Assembly
Contig A
Contig R
Contig B
Contig C
B
Contig or Scaffold file (FASTA)
Read file (FASTQ or FASTA)
Find all overlapping contigs
End to end De Bruijn search
Find edges using paired reads
Edge A→R found.
Remove duplicate edges.
Edges R→B and R→C found.
Edge R→B has been reported twice, duplication removed
Edges A→B R→B and B→ R found.
Untangle graph
Edge A→B is already described by path A→R→B and is removed.
Contig adjacency graph (CAG)
CAG can now be viewed using Contiguity
C
Contig A
Contig R
Contig B
Contig C

Figure 3: Flowchart of the graph creation process demonstrated using a mock genome assembly. A) Assembled contigs and their underlying reads. This region contains three unique contigs (A, B and C) separated by two repeat regions that have collapsed into a single contig (R). B) Flowchart of Contigutiy-CAG construction. Yellow notes show edges that were found and removed at each step if a *k*-mer size of 5 is used. Edges between contigs that overlap by more

than or equal to the *k*-mer size are found by searching for overlapping contig ends. Contigs that overlap by less than the *k*-mer size are found by finding paths through a De Bruijn graph. In some instances a path through the De Bruijn graph doesn't exist (e.g. between contigs R and C the nodes CCGTG and CGTGA do not exist in the read data) these edges are found by looking at mapped paired-end data. The edge between R and B is found by both paired-end and De Bruijn methods, this duplication of results is later removed. The edge between A and B is already described by the path A $\rightarrow$ R $\rightarrow$B, to reduce the complexity of the graph it is removed. Panel C shows the final construction of the graph illustrated in a manner similar to Contiguity.

The approaches utilized for Contiguity-CAG creation are summarized in more detail below:

i) Overlapping contig searches. Due to the nature of De Bruijn graphs, adjacent contigs assembled using a De Bruijn approach, such as Velvet or Spades, often overlap by one base pair less than the *k*-mer size used for assembly ($k - 1$). This enables adjacent contigs to be identified by looking for exact matches of length $\geq k - 1$ between contig ends. Generally, contig breaks are due to repetitive regions in the genome that cause branches in the De Bruijn graph. When the De Bruijn graph is resolved into contigs, assembly breaks occur where the graph branches. As the *k*-mers of adjacent nodes overlap by $k - 1$, the sequence of adjacent contigs based on those nodes will also overlap by the same amount. Other assemblers, such as Abyss, create contigs that can overlap by a much greater amount. To account for this possibility Contiguity also allows for large inexact overlaps.

ii) End-to-end De Bruijn search. An end-to-end De Bruijn search allows us to find adjacent contigs that do not overlap or overlap by an amount shorter than the defined *k*-mer value. This approach allows determination of the size of short overlaps between contigs or the sequence between contigs that don't overlap. Some regions of a De Bruijn graph assemble into contigs too short to be reported in the final assembly. This can result in unassembled segments of genome from a few, to hundreds of base pairs long. Although these regions of the genome are not covered by assembled sequence, they are covered by read data and as such can be reconstructed. These edges will only be found by paired-end reads if the unassembled segment is smaller than $i\text{-}2m$ where i is the insert size of the reads and m is the minimum length needed for the read to map to a contig (for global alignments this is usually close to the read length). Using a De Bruijn approach also allows the prediction of sequence between the contigs and identification of edges between adjacent contigs to distant to be resolved by paired-end reads. In our test data (see below) these unassembled segments rarely exceeded 300 bp in length and as this method uses a depth-limited search a default limit of 300bp plus the *k*-mer size allows the search to be performed in reasonable time by a desktop computer (Less than 10 minutes for an *E. coli* genome with ~200x coverage).

To find adjacencies between contigs separated by such regions, Contiguity constructs a De Bruijn graph from supplied read data using Khmer [24]. Khmer stores *k*-mer counts using a bloom filter allowing quick and memory efficient graph construction. Following construction of the graph, the frequency of each *k*-mer count is calculated. Two cutoffs (A and B) are generated from *k*-mer count frequencies: cutoff A is set as the first local minima of the *k*-mer count frequency graph, and cutoff B is set at half the first local maxima after cutoff A. Contiguity then performs a depth limited (default: 300 bp + *k*-mer size) search of the De Bruijn graph from the *k*-mer representing the 5’ and 3’ end of each contig. This search follows 3 rules: i) All *k*-mers with a frequency greater than cutoff B are considered to be real sequence and are traversed. ii) All *k*-mers with a frequency less than cutoff A are attributed to sequencing errors and are not traversed. iii) If the frequency of the *k*-mers in the De Bruijn graph are less than cutoff B and greater than cutoff A the less frequent *k*-mers are considered to be likely sequencing errors and only the most common *k*-mer is traversed (Figure 4). If the *k*-mer representing the 5’ or 3’ end of a contig is found during the search an edge is created between the two contigs. If more than one path is found between two contigs the path with the highest *k*-mer coverage chosen as the correct path.

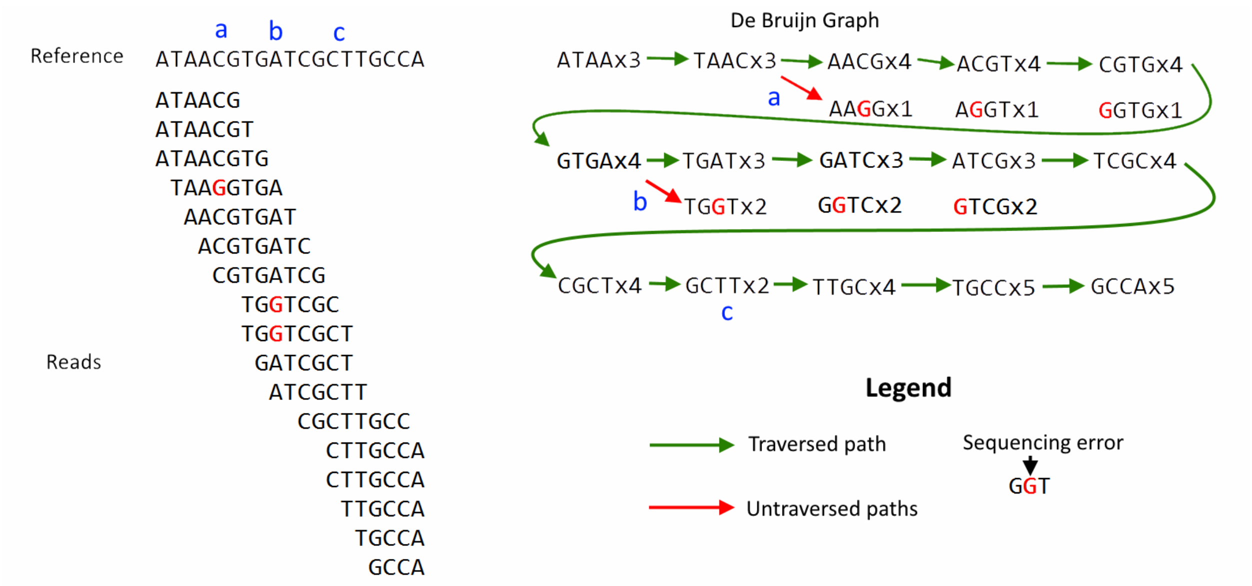


Figure 4: Traversal of the De Bruijn graph. Representation of a reference sequence, reads and the De Bruijn Graph. Sequencing errors are shown with a red base. The green arrows shows the path traversed from the first node to the last node when cutoff A is set to 2 and cutoff B is set to 4. Red arrows show paths considered but not traversed. a) A sequencing error found in a single read, this path is not traversed because it falls under cutoff A. b) A sequencing error found in multiple reads. This path is not traversed because there is a more likely path forward. c) A low coverage region. The De Bruijn search algorithm used by Contiguity will search low coverage nodes if no alternative path is found.

iii) Paired-end search. Reads are aligned using Bowtie 2 [25] and edges are created between contigs that a) share more than a user-defined number of paired reads, b) are orientated in the correct direction and c) are situated at the end of the contig. Sequencing specific error profiles often create a coverage profile where reads overlap by an amount shorter than the *k*-mer used for assembly. These regions of the graph cannot be traversed during *De Bruijn* assembly and result in a contig break (if additional scaffolding is not performed). If adjacent contigs found using this method have exact overlapping ends of at least 5 base pairs (probability of identical bases due to chance < 0.001) and paired-end data indicates that the contigs should overlap, the contigs were likely split due to a sequence specific error profile and Contiguity considers them to be overlapping. If there is no detectable overlap, user-defined scaffolding characters (such as "NNNNN") are inserted between the two contigs that have paired-end scaffold support.

Removal of duplicate and redundant edges. Where two or more overlapping edges are found between the same contig ends the larger overlap is preferred. In the event where both an overlapping edge and a non-overlapping edge are found between the same contig ends, the overlapping edge is chosen. Contiguity also removes redundant edges using an algorithm called untangle (Figure 3).

Manual comparison of Contiguity, Velvet and Abyss CAGs

To verify the contig adjacency graph and provide a comparison to currently used methods such as traversal of Velvet's LastGraph and Abyss Assembler's .dot graph file a sequencing run of *Escherichia coli* str. UTI89 (ENA accession: ERR687901) was assembled and examined using the Contiguity GUI. Genomic DNA from *E. coli* str. UTI89 was sequenced using Illumina HiSeq2000. Reads with ends marked as low quality (phred quality score ≤ 2) were trimmed and reads were filtered if the average per-base quality was less than 30 or had one pair trimmed to less than 50 base pairs. In total, 2,433,934 high quality read pairs with an average insert size of 367.25 base pairs with a standard deviation of 59.1 were assembled using i) Velvet (with scaffolding), ii) Velvet (without scaffolding), or iii) Abyss (with default parameters, with scaffolding). Insert size and standard deviation were provided to Velvet with the rest of the parameters detected automatically or left at default and *k*-mer size for all assemblies were chosen to optimize N50. Contiguity was then used to construct a CAG for each of the three assembly methods.

To determine the true adjacencies of the assembled sequence, contigs from the Velvet assembly were then mapped to the published UTI89 genome and the contigs that aligned to the first 250,000 base pairs and their CAGs were compared. This region was chosen because it contained a mix of long and short unique sequence and repetitive elements (Table 1). Contigs that aligned to the reference, with BLAST, along their full length with an identity of at least 95% were considered mapped to that region or regions. Velvet assembled this region into 26 contigs from unique regions and 12 collapsed repeats. Velvet's scaffolding reduced the amount of unique contigs to 11, but did not improve the assembly of the repetitive region.

To calculate precision and sensitivity of the Contiguity-CAG, all edges from the contigs mapped to the selected region were examined. This included edges to contigs mapping outside of the region. True and false positives were calculated using the following rules.

i. Edges between adjacent nodes were counted as true positives.
ii. Edges between non-adjacent nodes were counted as false positives.
iii. Adjacent contigs, within the examined region, that did not share an edge were counted as false negatives.

Velvet's LastGraph file consists of nodes and arcs between adjacent nodes. Without scaffolding, nodes larger than a user defined amount are reported in the final assembly as contigs. If paired-end data is made available, Velvet will join multiple nodes together to form a scaffold. As not all nodes are reported in the final assembly, edges between contigs were reconstructed from the LastGraph file. To accomplish this paths between contigs were found through manual exploration of the LastGraph using the Contiguity GUI. Edges were reconstructed using the following rules.

i. For all contigs or scaffolds reported in the final assembly the corresponding node or nodes were found in the LastGraph.
ii. If the contig or scaffold corresponded to a single node all arcs from that node were explored.
iii. If the scaffold consisted of multiple nodes only arcs from the end of the node that corresponded to the end of the scaffold were explored.
iv. Exhaustive searches of all paths starting at the identified arcs were performed. If a node corresponding to a contig or scaffold was found an edge is created between the two contigs or scaffolds.

As the LastGraph is constructed from the De Bruijn graph and does not utilize paired-end information the Contiguity-CAG was able to find significantly more edges than the Velvet-CAG. Combining scaffolding and LastGraph information improves the proportion of edges found however this method still fails to resolve a number of edges in the repetitive regions (Table 1).

Precision and sensitivity were then calculated for the Abyss-CAG using the same method used for the Contiguity-CAG. Unlike Velvet, Abyss attempts to scaffold fragmented repetitive regions and as such, a larger region needed to be chosen to evaluate the same number of edges. Contigs mapping to the first 1,500,000 base pairs and their edges were evaluated for both the Contiguity-CAG and Abyss-CAG (Table 1). Interval levels were calculated using the Adjusted-Wald method using a confidence level of 95%. 100% sensitivity was achieved in all three Contiguity graphs, to account for this sensitivity was estimated using the LaPlace method.

Table 1: Comparison of Contiguity, Velvet and Abyss CAGs.

Assembler	Scaffolding	CAG type	Contigs	Nodes	Unique Contigs		Contigs of Collapsed Repeats		Edges		
					Number	Average Length	Number	Average Length	Number	Precision	Sensitivity
Velvet	Yes	Contiguity	23	46	11	25515.5	12	497.8	78	78.2 ± 9.1	97.8 ± 4.0
Velvet	Yes	LastGraph	23	52	11	25515.5	12	497.8	55	81.8 ± 10.2	72.7 ± 12.87
Velvet	No	Contiguity	38	76	26	9825.8	12	497.8	109	81.7 ± 7.3	98.68 ± 2.5
Velvet	No	LastGraph	38	82	26	9825.8	12	497.8	55	83.3 ± 10.1	56.76 ± 11.0
Abyss	Yes	Contiguity	36	72	25	63430.7	11	1928.9	150	68.2 ± 7.4	98.61 ± 2.6
Abyss	Yes	Abyss	36	72	25	63430.7	11	1928.9	134	67.9 ± 10.1	94.3 ±6.2

Contiguity was able to find significantly more connections between contigs than traversal of Velvet's LastGraph with comparable precision. This was largely due to Velvet's not reporting paired-end information in the LastGraph and *k*-mer sizes and cutoffs chosen to optimize assembly and not graph construction. Sensitivity and precision of the Contiguity and Abyss CAGs were largely comparable, although Abyss failed to find 2 edges between adjacent contigs. In both these cases one of the contigs was a short (<110bp) collapsed repeat.

Automated comparison of Contiguity

To further assess the quality of the contig adjacency graphs produced by Contiguity, and identify potential limitations of this method, Contiguity-CAGs were generated for *de novo* assemblies of 33 test datasets in which both unassembled reads and a complete genome sequence were available (Table 2). The test datasets represent a variety of bacterial genomes with examples of both simulated and real Illumina HiSeq paired-end data. GemSim (v1.6) [26] was used to simulate Illumina HiSeq 2000 reads at 100-120× coverage for each of 25 bacterial genomes. Furthermore, 5 Illumina HiSeq II, and 1

Illumina Hiseq IIx datasets, that are available via the short read archive (SRA) and also had available complete genome sequences, were tested. Two independent HiSeq 2000 genome sequencing runs from the reference *E. coli* strain UTI89 previously mentioned were also included (acc: PRJEB7805). Reads from the simulated datasets had a length of 101bp while real lengths in the real dataset ranged from 76bp to 101bp. Insert sizes ranged from 234 to 358. All genomes were assembled using Velvet (v1.2.07), *k*-mer size was chosen to optimize the N50 value of the assembly and insert size was provided. All other paramaters were detected automatically. Scaffolding was turned off to reduce inaccuracy in mapping assembled contigs or scaffolds back to the reference. All other parameters were detected automatically by Velvet.

For each assembly, a contig adjacency graph was created using Contiguity and default parameters (Contiguity-CAG). Because of the large volume of data being analysed an in-house script (available at https://github.com/mjsull/Contiguity) was needed to estimate the precision and sensitivity of our graphs. All sequence comparisons were performed using BLAST, the full length of the contig must align with at least 95% identity for the contig to be considered "mapped". The precision and sensitivity of the edges created for each assembly were tested using the following rules:

- **i.** For each edge found between two contigs, A and B, if both contig A and B are mapped to the reference sequence (and therefore aren't misassembled) a scaffold of contig A and B is created.
- **ii.** If the scaffold of contig A and B mapped to the reference the edge is a true positive.
- **iii.** If the scaffold is not found then the edge is a false positive.

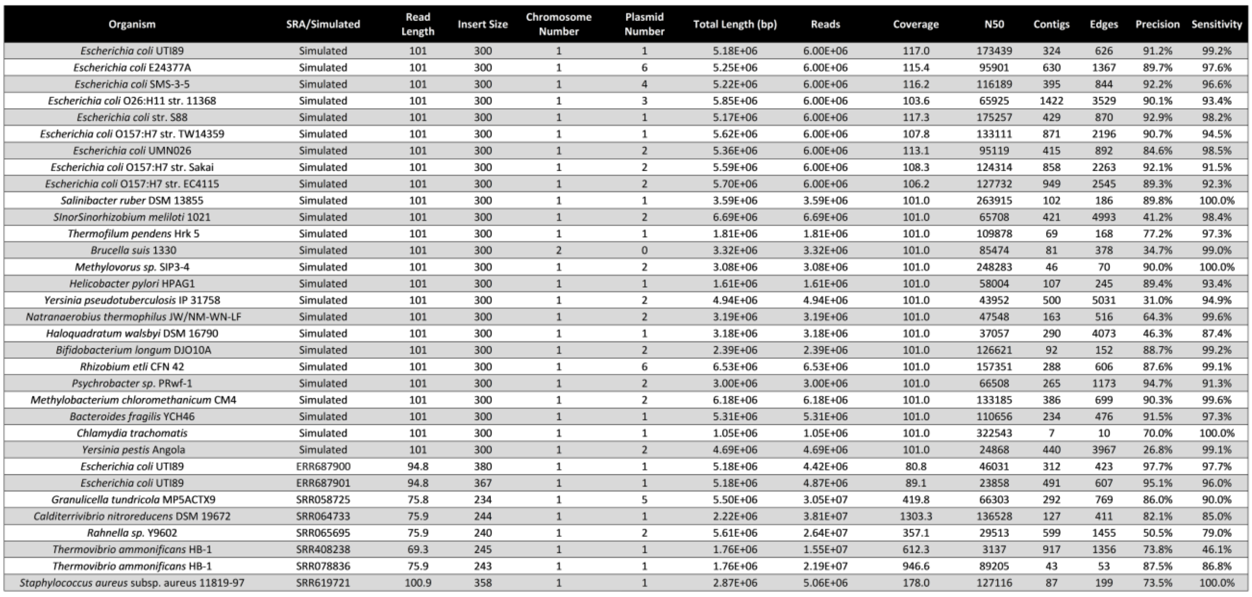
The script then creates a list of ordered contigs by mapping them to the reference using the following rules.

- **i.** If a contig maps within an area a larger contig maps to it is not included in the list.
- **ii.** If two contigs map to the same area the contig with a lower identity is removed from the list.
- **iii.** Contigs may be placed more than once.

The script then traverses through the ordered list of contigs, if an edge exists between two adjacent contigs with the correct orientation it is considered a true positive for the purpose of calculating sensitivity, if no edge exists it is considered a false negative.

Repetitive sequence often assembles into multiple redundant contigs. As a result, the repetitive regions of a genome can often be accurately reconstructed from contigs in multiple ways. Contigs automatically chosen to represent a region sometimes differ from what would be chosen manually, although both accurately represent the underlying genomic sequence. As the script only checks to see if edges between automatically selected contigs exists, and not if a repetitive region could be accurately reconstructed using alternative contigs, the predicted sensitivity tends to be lower than the real value (Table 2).

Table 2: Estimates of precision and sensitivity of Contiguity across multiple strains.



Organism	SRA/Simulated	Read Length	Insert Size	Chromosome Number	Plasmid Number	Total Length (bp)	Reads	Coverage	N50	Contigs	Edges	Precision	Sensitivity
Escherichia coli UTI89	Simulated	101	300	1	1	5.18E+06	6.00E+06	117.0	173439	324	626	91.2%	99.2%
Escherichia coli E24377A	Simulated	101	300	1	6	5.25E+06	6.00E+06	115.4	95901	630	1367	89.7%	97.6%
Escherichia coli SMS-3-5	Simulated	101	300	1	4	5.22E+06	6.00E+06	116.2	116189	395	844	92.2%	96.6%
Escherichia coli O26:H11 str. 11368	Simulated	101	300	1	3	5.85E+06	6.00E+06	103.6	65925	1422	3529	90.1%	93.4%
Escherichia coli str. S88	Simulated	101	300	1	1	5.17E+06	6.00E+06	117.3	175257	429	870	92.9%	98.2%
Escherichia coli O157:H7 str. TW14359	Simulated	101	300	1	1	5.62E+06	6.00E+06	107.8	133111	871	2196	90.7%	94.5%
Escherichia coli UMN026	Simulated	101	300	1	2	5.36E+06	6.00E+06	113.1	95119	415	892	84.6%	98.5%
Escherichia coli O157:H7 str. Sakai	Simulated	101	300	1	2	5.59E+06	6.00E+06	108.3	124314	858	2263	92.1%	91.5%
Escherichia coli O157:H7 str. EC4115	Simulated	101	300	1	2	5.70E+06	6.00E+06	106.2	127732	949	2545	89.3%	92.3%
Salinibacter ruber DSM 13855	Simulated	101	300	1	1	3.59E+06	3.59E+06	101.0	263915	102	186	89.8%	100.0%
SlnorSinorhizobium meliloti 1021	Simulated	101	300	1	2	6.69E+06	6.69E+06	101.0	65708	421	4993	41.2%	98.4%
Thermofilum pendens Hrk 5	Simulated	101	300	1	1	1.81E+06	1.81E+06	101.0	109878	69	168	77.2%	97.3%
Brucella suis 1330	Simulated	101	300	2	0	3.32E+06	3.32E+06	101.0	85474	81	378	34.7%	99.0%
Methylovorus sp. SIP3-4	Simulated	101	300	1	2	3.08E+06	3.08E+06	101.0	248283	46	70	90.0%	100.0%
Helicobacter pylori HPAG1	Simulated	101	300	1	1	1.61E+06	1.61E+06	101.0	58004	107	245	89.4%	93.4%
Yersinia pseudotuberculosis IP 31758	Simulated	101	300	1	2	4.94E+06	4.94E+06	101.0	43952	500	5031	31.0%	94.9%
Natranaerobius thermophilus JW/NM-WN-LF	Simulated	101	300	1	2	3.19E+06	3.19E+06	101.0	47548	163	516	64.3%	99.6%
Haloquadratum walsbyi DSM 16790	Simulated	101	300	1	1	3.18E+06	3.18E+06	101.0	37057	290	4073	46.3%	87.4%
Bifidobacterium longum DJO10A	Simulated	101	300	1	2	2.39E+06	2.39E+06	101.0	126621	92	152	88.7%	99.2%
Rhizobium etli CFN 42	Simulated	101	300	1	6	6.53E+06	6.53E+06	101.0	157351	288	606	87.6%	99.1%
Psychrobacter sp. PRwf-1	Simulated	101	300	1	2	3.00E+06	3.00E+06	101.0	66508	265	1173	94.7%	91.3%
Methylobacterium chloromethanicum CM4	Simulated	101	300	1	2	6.18E+06	6.18E+06	101.0	133185	386	699	90.3%	99.6%
Bacteroides fragilis YCH46	Simulated	101	300	1	1	5.31E+06	5.31E+06	101.0	110656	234	476	91.5%	97.3%
Chlamydia trachomatis	Simulated	101	300	1	1	1.05E+06	1.05E+06	101.0	322543	7	10	70.0%	100.0%
Yersinia pestis Angola	Simulated	101	300	1	2	4.69E+06	4.69E+06	101.0	24868	440	3967	26.8%	99.1%
Escherichia coli UTI89	ERR687900	94.8	380	1	1	5.18E+06	4.42E+06	80.8	46031	312	423	97.7%	97.7%
Escherichia coli UTI89	ERR687901	94.8	367	1	1	5.18E+06	4.87E+06	89.1	23858	491	607	95.1%	96.0%
Granulicella tundricola MP5ACTX9	SRR058725	75.8	234	1	5	5.50E+06	3.05E+07	419.8	66303	292	769	86.0%	90.0%
Calditerrivibrio nitroreducens DSM 19672	SRR064733	75.9	244	1	1	2.22E+06	3.81E+07	1303.3	136528	127	411	82.1%	85.0%
Rahnella sp. Y9602	SRR065695	75.9	240	1	2	5.61E+06	2.64E+07	357.1	29513	599	1455	50.5%	79.0%
Thermovibrio ammonificans HB-1	SRR408238	69.3	245	1	1	1.76E+06	1.55E+07	612.3	3137	917	1356	73.8%	46.1%
Thermovibrio ammonificans HB-1	SRR078836	75.9	243	1	1	1.76E+06	2.19E+07	946.6	89205	43	53	87.5%	86.8%
Staphylococcus aureus subsp. aureus 11819-97	SRR619721	100.9	358	1	1	2.87E+06	5.06E+06	178.0	127116	87	199	73.5%	100.0%

Contiguity was able to achieve high precision and sensitivity across a large range of bacterial genomes and read coverage (Figure 5). In the case of *Thermovibrio ammonifacans* HB-1 (Table 2), two independent sequencing runs gave very different assembly metrics (917 and 43 contigs, respectively), despite good read coverage being achieved in both runs (612x and 946x, respectively). Predicted sensitivity for the *T. ammonifacans* HB-1 sequencing run that assembled into 917 contigs was 46.1%, compared to 86.8% for the independent 43 contig assembly. All real datasets from genomes sequenced with newer platforms, such as the HiSeq IIx and HiSeq 2000 resulted in higher sensitivity graphs than those sequenced with older platforms, such as the Genome Analyser II (100%, 97.7% and 96.0% vs. 90.0%, 86.8%, 85.0%, 79.0% and 46.1%).

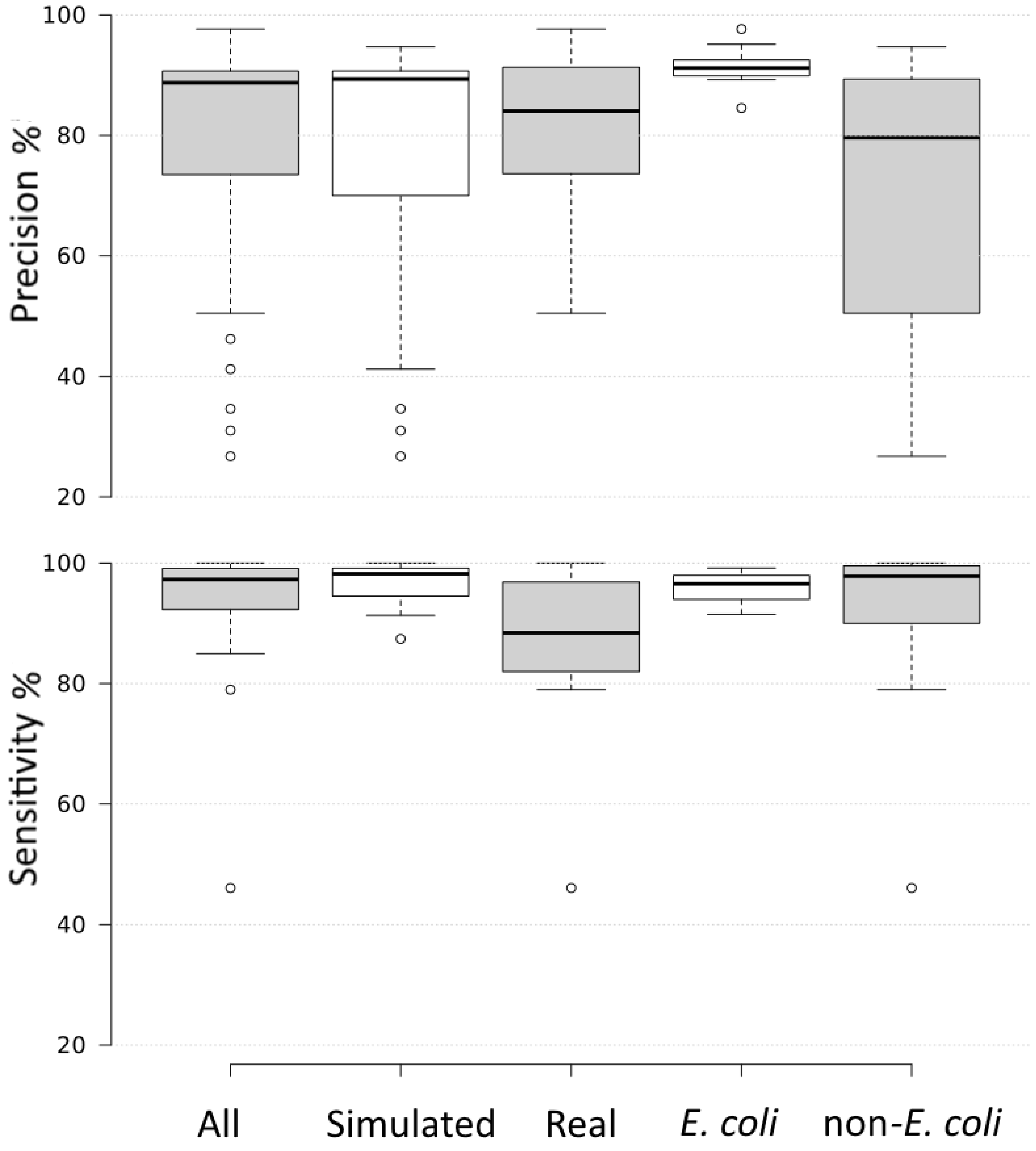


Figure 5: Precision and sensitivity of Contiguity-CAGs. Precision and sensitivity are calculated from the number of True positives (TP), false positives (FP) and false negatives (FM). Precision is calculated as TP/(TP+FP), sensitivity is calculated as TP/(TP+FN). Graph also shows a

comparison between real and simulated sequencing and assemblies of *E. coli* and non-*E. coli* genomes.

Precision seems to be less dependent on sequencing and assembly quality and more on the composition of repetitive elements of the genome and how they reconstructed by the assembler. Precision only weakly correlates to assembly quality (*r*=0.338, using N50 estimate assembly quality). Precision correlates more strongly to the percent of total contigs in an assembly that are repetitive (*r*=0.599, repetitive contigs defined as contigs that map more than once to the reference). One reason for this is incorporation of repetitive sequence into the ends of unique contigs resulting in fewer repetitive contigs and lower precision (Figure 6). Lower variation and higher median precision was observed in *E. coli* assemblies (Figure 5), this also corresponded to a higher percentage of repeat contigs with lower variation (62.3% average with a standard deviation of 4.7% in *E. coli* compared to 39.2% average with a standard deviation of 14.5% in non-*E. coli*).

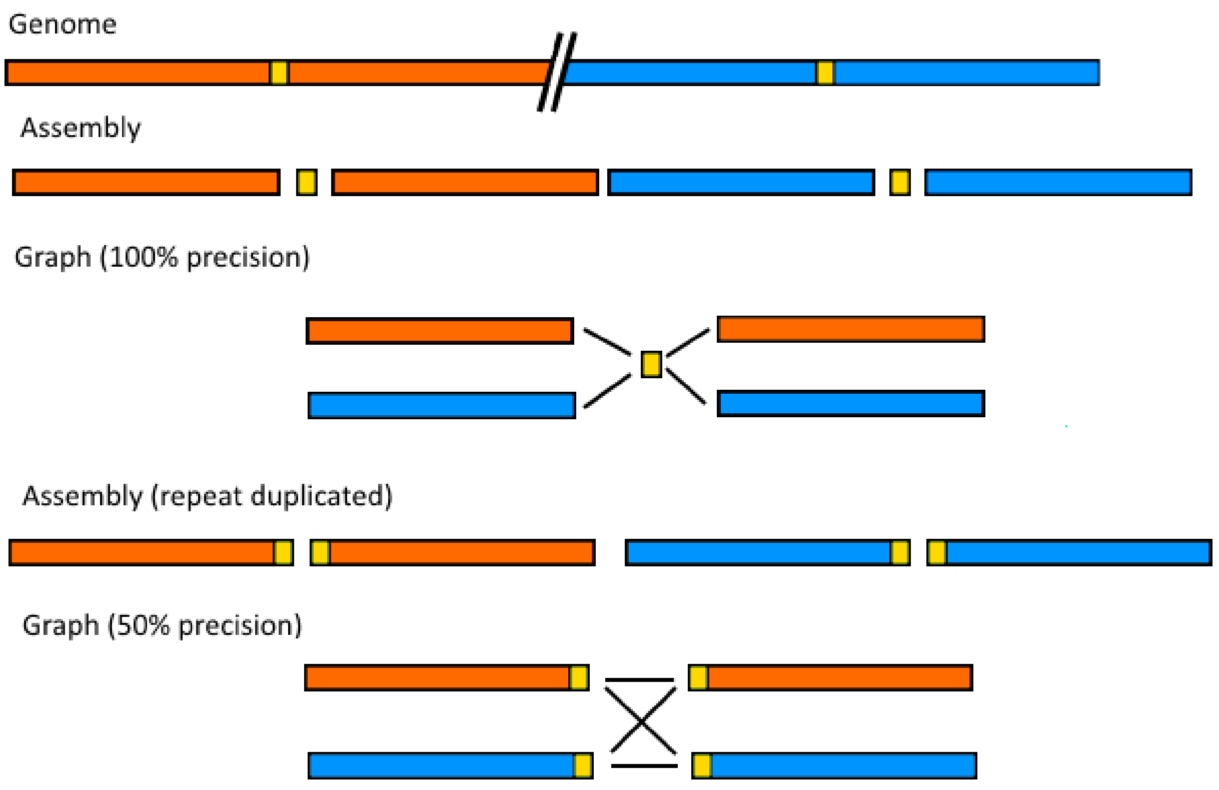


Figure 6: Illustration of a repetitive region of the genome, how it assembles and the resulting graph. Incorporation of repetitive regions (yellow) of chromosome into adjacent contigs results

in lower precision. In the first assembly the repeat region assembles into its own contig, this results in a 100% precision in the graph. In the second assembly the repeat region is incorporated into the ends of adjacent contigs. Because the repeat is too large to resolve, the resulting CAG has lower precision. The two CAGs are only superficially different, despite the difference in precision as both graphs can be arranged in 2 possible ways.

When provided with high-quality data Contiguity is able to produce highly sensitive adjacency graphs. A high sensitivity is essential for identifying and reconstructing large novel sections of a genome.

Case studies

Case study 1: PacBio SMRT sequencing closure

***DATA**: E. coli* str. EC958 is a representative sequence type 131 (ST131) O25b strain of uropathogenic *E. coli* which is characterized by several prophage and genomic island regions as well as a large antibiotic resistance plasmid [27]. EC958 genomic DNA was sequenced on the PacBio RS I instrument generating a total of 601,224 pre-filtered reads with an average length of 1,600 bp, from six SMRT cells [28]. Reads were assembled *de novo* using the hierarchical genome assembly process (HGAP) from the PacBio SMRT analysis package (V2.0.0)[2] with default settings and a seed read cut-off length of 5,000 base-pairs (bp). A total of 7 contigs were generated from the initial HGAP assembly, all contigs were shorter than the expected chromosome size.

AIM i): Close the genome of *E. coli* EC958 assembled with SMRT sequencing reads.

WORKFLOW AND RESULTS i):

1. Load FASTA of contigs into Contiguity.
2. Generate a self-comparison from within Contiguity using the NCBI-BLAST+ (BLASTn).

Overlapping but un-joined contigs are a characterized artefact of the HGAP assembly process [2] and in the EC958 assembly several overlapping edges were evident between contigs (Figure 7A).

3. Change the size of all contigs in Contiguity from relative (node size proportionate to length of contig in base pairs) to constant (node size identical for all contigs).

4. Change self-comparison so that only BLASTn hits between overlapping edges are shown.

A circular chromosome and large ~140 kb plasmid are identifiable (Figure 7B).

AIM ii): Investigate the 12,666bp, 12,206bp and 8483bp linear contigs in the HGAP assembly by viewing all BLASTn comparisons between (but not within) contigs.

WORKFLOW AND RESULTS ii):

1. Change self-comparison so that only BLASTn hits between contigs are shown.

The entire sequence of all three contigs was found in the three large contigs that make up the chromosome of EC958. Alignments indicated small inverted regions in the three contigs under investigation (Figure 7C). Subsequent similarity searches indicated that these ~3 kb inversions corresponded to DNA invertase mediated prophage tail fibre allele switching within a high proportion of the *E. coli* cells that were grown prior to DNA harvesting and library preparation [28]. This phenomenon has long been recognized as a mechanism for altering host specificity of bacteriophage by alternating in-frame C-terminal phage-tail protein fragments [29]. This phenomena is generally not readily identifiable in draft genomes generated with shorter-read technologies without mate-pair libraries.

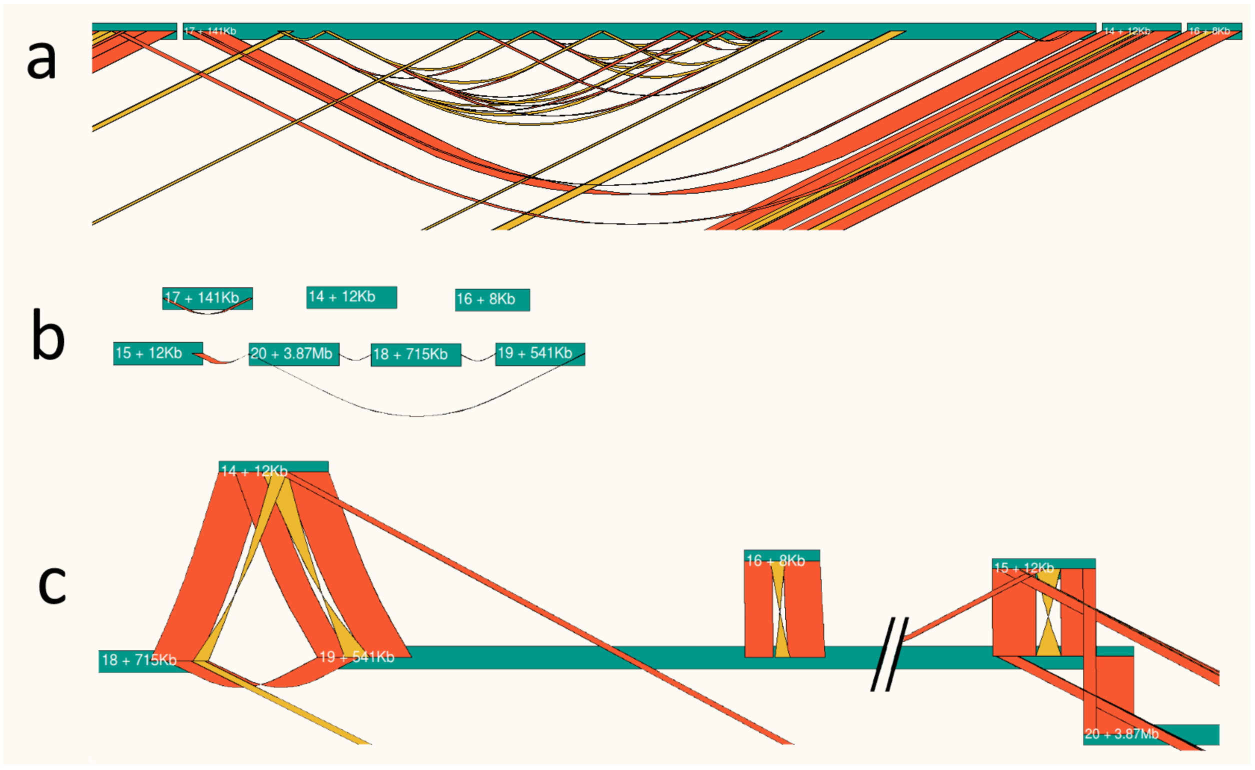


Figure 7: Pacbio assembly analysis using Contiguity. A) Contiguity representation of a small section of the entire PacBio assembly of *E. coli* str. EC958. Contigs are labeled with their name, orientation and length. BLAST hits satisfying user defined parameters are shown as ribbons. Direct repeats are shown in orange and inverted repeats are shown in yellow. B) A Contiguity representation of the 5.1Mbp circularised genome of *E. coli* str. EC958 with a uniform size for all contigs to enable visualization of an entire assembly in a single view. The main chromosome is broken into three large contigs: 18, 19 and 20. Contig 17 is a 141Kb circular plasmid as indicated by direct repeats linking the 5' and 3' ends. Contigs 14, 15 and 16 are 8-12 Kb linear segments of DNA. C) A self-comparison of all assembled contigs rearranged using the Contiguity interactive browser to show three inversion events within the genome of EC958. These inversions seemed to be the cause of at least two contig breaks. Similarities between the inverted region of contig 14 and 15 were also found (not shown).

2. Create scaffolds of the chromosome and plasmid using the "Write FASTA" tool.

Once the cause of the fragmentation of the genome was identified, the three long chromosomal contigs can be scaffolded into a single contig and the short contigs containing the alternate phage tail gene arrangement were removed. These regions can later be annotated as regions of variation in the final assembly. Contiguity can automatically remove overlapping sequence on the end of each contig if instructed by the user, allowing the user to create a single unbroken scaffold for both the chromosome and plasmid. This duplication of sequence is usually caused by the HGAP assembly process, however, overlapping contigs may be caused by other phenomena, such as long repetitive regions. Scaffolds created using Contiguity should be verified by mapping reads back onto the newly created sequence.

Case Study 2: Reconstruction of prophage and plasmid sequence in a draft assembly

DATA: E. coli. str. UTI89 is a well characterized uropathogenic bacteria often used in mouse models to study urinary tract infections [30]. A comparative and graphical analysis of the UTI89 Illumina sequencing data assembled with Velvet (described previously) was performed.

AIM i): reconstruction of novel chromosomal regions in *E. coli* str. UTI89 draft assembly.

WORKFLOW AND RESULTS i):

1. *Generate* a contig adjacency graph was generated using Contiguity's CAG creation tool.
2. Load the resulting graph.
3. To aid interpretation of the assembly, use Contiguity to launch a BLASTn comparison against *E. coli* UM146, UTI89's nearest neighbour with a finished genome.
4. Display contigs mapping to UM146.

An insertion relative to UM146 was identified in UTI89 between contigs 45 and 217 as indicated by a break in the contig adjacency chain displayed in Contiguity (Figure 8A). As there is no read evidence suggesting that contigs 45 and 217 are adjacent it can be concluded that additional contigs, that were not displayed as they didn't map to the reference chromosome, may be part of this insertion.

5. Use the "Find Paths" tool to find all paths between contigs 45 and 217

Using Contiguity, a single path of two contigs was found between contig 45 and 217 with the "Find Paths" tool, a modified depth-limited search that finds the all paths in the adjacency graph between two contigs (Figure 8B).

6. Recreate the prophage at this site using the "Write FASTA" tool.

Further analysis of this contig can be achieved by analysing the sequence outside of Contiguity (Figure 8C). It is worth noting that a simple region such as this, where only a single path exists between the two contigs, could also be reconstructed using scaffolding software. Contiguity's "Find Paths" tool is able to report all paths between two contigs, this allows insight into potential arrangements of regions where ambiguity exists. This can be incredibly useful for reconstructing novel regions of the chromosome where traditional methods, such as ordering to a reference or scaffolding will fail.

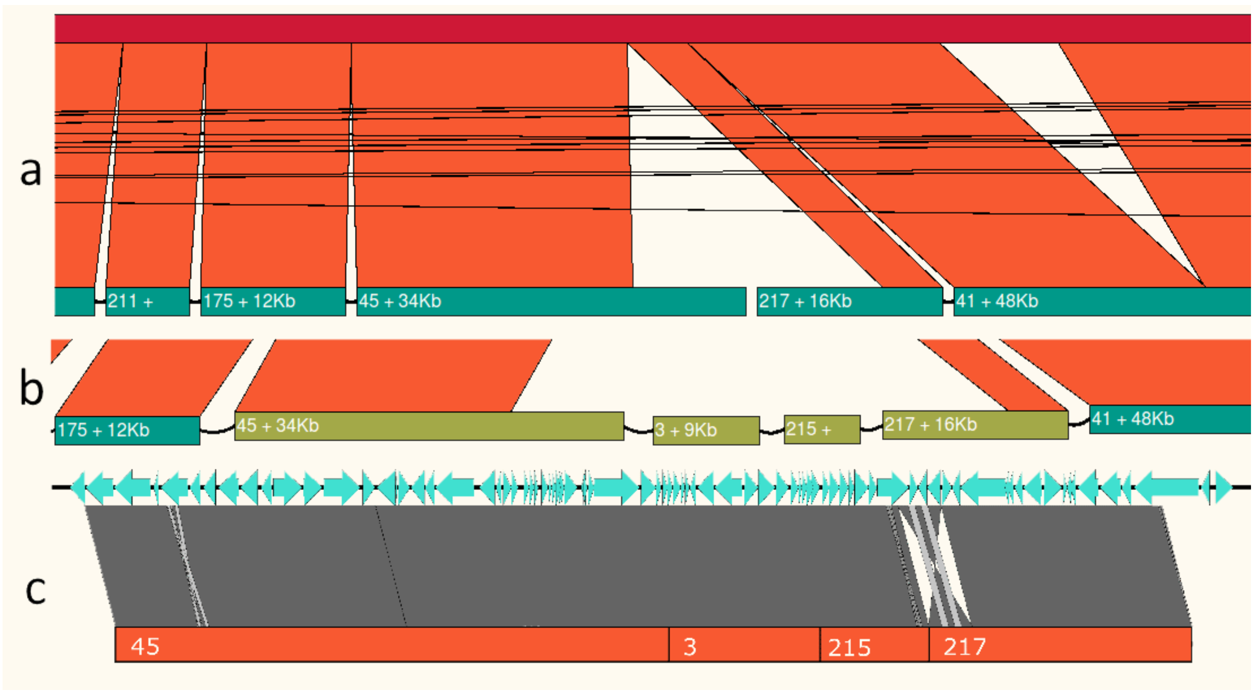


Figure 8: Reconstruction of a prophage using Contiguity. A) A small section of the Illumina assembly (dark cyan) aligned against UM146 (red). BLASTn hits are shown in orange. Arcs between contigs represent contig adjacencies. B) Putative contig order at insertion site found using the "Find paths" tool. Insertion site in UTI89 contains several putative phage genes. C) Scaffold created with Contiguity compared back to the published UTI89 genome. A small inversion, possibly due to DNA invertase mediated prophage tail fibre allele switching, has occurred in our isolate. Panel C was created with Easyfig.

AIM ii): reconstruction of novel plasmid sequence in *E. coli* str. UTI89 draft assembly.

WORKFLOW AND RESULTS ii):

Although 359 contigs of the UTI89 assembly have at least 100bp of their sequence aligned to the UM146 chromosome with at least 95% identity, a large number of contigs in the UTI89 assembly have no alignment to UM146 and further inspection of these contigs reveal no obvious insertion sites in the draft chromosome. The interconnectivity of these contigs suggest that they are assembled from the same region of the genome, consistent with a mobile genetic element such as a plasmid, genomic island or prophage.

1. Display UTI89 contigs with no BLASTn hits to UM146 longer than 100bp using the "View Assembly" function (Figure 9A).
2. Use the "Find Paths" tool to reinsert regions contigs of sequence shared by the chromosome and plasmid

Removing all UTI89 contigs with sequence similarity to UM146 potentially removed repetitive regions shared by both the UTI89 extrachromosomal element(s) and the chromosome. Therefore, to reintroduce these repetitive contigs the "Find Paths" function in Contiguity was used to find all paths less than 3000 bp between the selected contigs. This value should be large enough to span insertion elements found in the assembly, but small enough to avoid finding spurious paths. Of the 25 novel contigs identified as non-UM146 with the "View assembly" function, and 8 collapsed repeat contigs added with the "Find paths" function, 30 contigs create a circuit in the subgraph. It is possible to infer that because of their circular nature that these 30 contigs comprise a large plasmid approximately 115 kb in length.

3. Order contigs in the circuit according to the Contiguity-CAG graph.

To do this each contig was put adjacent to another contig if there is read evidence that they are adjacent (Figure 9B). Contigs of collapsed repeat regions are duplicated where necessary. The number of repeat regions in the plasmid can be predicted by identifying the number of contigs the repeat is adjacent to. A large (~115Kbp) circular region has now been identified with four possible arrangements (Figure 9C).

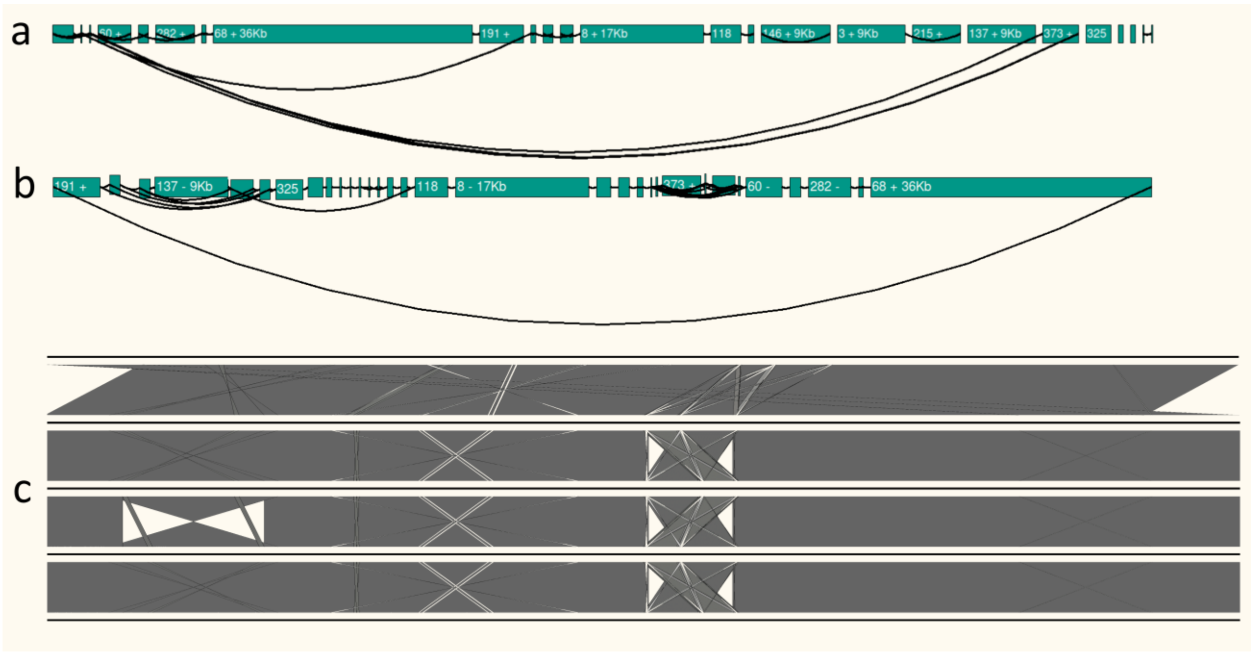


Figure 9: Identification and ordering of plasmid contigs. a) Contigs with no BLASTn hits to the reference, the interconnectivity of the subgraph suggest the majority of these contigs may be part of a single, large mobile genetic element. b) Contigs of repetitive regions are added using the "Find paths" tool, contigs are then ordered into one of four possible arrangements using the graph. c) The four possible arrangements of the contigs identified and ordered using Contiguity compared to the published UTI89 plasmid (top).

Primers can then be designed to confirm the presence of the plasmid and order of the contigs. However in this case, as the reference sequence is available, the accuracy of this method can be illustrated by comparing the 4 potential plasmid scaffolds back to the pUTI89 reference. If unique contigs were ordered blindly, the number of possible arrangements would be $\prod_{k=1}^{n-1} 2k$, where n is the number of unique contigs. 1.08×10^{26} for the 22 unique contigs found in this plasmid. Adding repeat contigs would increase this number by an order of magnitude. Contiguity creates and presents adjacency information in an intuitive manner allowing the plasmid and potential orders to be easily visualized and reconstructed.

Conclusions

Contiguity includes the first sequencing and assembly independent method of producing contig adjacency graphs. Adjacency graphs, when combined with comparative genomics, can be a powerful tool for identifying and reconstructing large novel sections of a genome.

A purpose built graph construction algorithm improves upon current graph creation methods in several ways. Firstly, the ability to choose a smaller *k*-mer for graph creation than used during assembly allows lower coverage regions between contigs to be traversed. Secondly, adopting a De Bruijn approach allows us to traverse regions of the genome larger than commonly used insert sizes. Consequently, connections between contigs separated by gaps larger than the insert-size of paired reads are identifiable. When combined with a paired read approach, the result is an extremely sensitive contig adjacency graph.

Contiguity provides an easy to use graphical user interface with a large amount of functionality. A linear layout with edges represented as arcs makes exploring an assembly as a graph more intuitive without necessarily cluttering the graph [31]. Contiguity also enables easy visualization of contig metadata, such as coverage or sequence composition. The layout and features contained within the software package allow quick and easy analysis *de novo* assemblies and their graphs.

The case studies chosen illustrate two possible uses of Contiguity in the analysis of *de novo* assemblies that were previously only possible using multiple applications, scripts or manual inspection. These examples show how it is easy to identify and provide context, such as synteny, to regions of the genome that are difficult to assemble including prophages and plasmids. They also demonstrate how easily contigs can be ordered and scaffolded for further analysis.

In future, improvements to Contiguity will allow it to be used with larger eukaryotic or metagenomic datasets. Moving infrequently accessed data from memory to disk would allow larger CAGs to be visualized with less memory. Currently, all sequence alignments are shown, this results in some alignments between the reference and graph, which are out of frame, crisscrossing the screen. Only showing alignments for the region of the reference, or contigs, in frame would make graphs clearer and require less processing power to render comparisons. Contiguity will be adapted to both produce and read GFA (Graphical Fragment Assembly) files once the format has been formally described.

Acknowledgments

The authors would also like to thank Professor Mark Schembri for making available the UTI89 Illumina datasets used in this project.

Funding

SAB is supported by a Career Development Fellowship from the National Health and Medical Research Council Career of Australia [grant number APP1090456].

References

1. Chain PS, Grafham DV, Fulton RS, Fitzgerald MG, Hostetler J, Muzny D, Ali J, Birren B, Bruce DC, Buhay C *et al*: **Genomics. Genome project standards in a new era of sequencing**. *Science (New York, NY)* 2009, **326**(5950):236-237.
2. Chin CS, Alexander DH, Marks P, Klammer AA, Drake J, Heiner C, Clum A, Copeland A, Huddleston J, Eichler EE *et al*: **Nonhybrid, finished microbial genome assemblies from long-read SMRT sequencing data**. *Nature methods* 2013, **10**(6):563-569.
3. Sullivan MJ, Petty NK, Beatson SA: **Easyfig: a genome comparison visualiser**. *Bioinformatics (Oxford, England)* 2011.
4. Carver TJ, Rutherford KM, Berriman M, Rajandream MA, Barrell BG, Parkhill J: **ACT: the Artemis Comparison Tool**. *Bioinformatics (Oxford, England)* 2005, **21**(16):3422-3423.
5. Guy L, Kultima JR, Andersson SG: **genoPlotR: comparative gene and genome visualization in R**. *Bioinformatics (Oxford, England)* 2010, **26**(18):2334-2335.
6. Thorvaldsdóttir H, Robinson JT, Mesirov JP: **Integrative Genomics Viewer (IGV): high-performance genomics data visualization and exploration**. *Briefings in bioinformatics* 2013, **14**(2):178-192.
7. Darling AC, Mau B, Blattner FR, Perna NT: **Mauve: multiple alignment of conserved genomic sequence with rearrangements**. *Genome Res* 2004, **14**(7):1394-1403.

8. Hayashi T, Makino K, Ohnishi M, Kurokawa K, Ishii K, Yokoyama K, Han CG, Ohtsubo E, Nakayama K, Murata T *et al*: **Complete genome sequence of enterohemorrhagic *Escherichia coli* O157 : H7 and genomic comparison with a laboratory strain K-12**. *DNA Research* 2001, **8**(1):11-22.

9. Lanza VF, de Toro M, Garcillan-Barcia MP, Mora A, Blanco J, Coque TM, de la Cruz F: **Plasmid flux in *Escherichia coli* ST131 sublineages, analyzed by plasmid constellation network (PLACNET), a new method for plasmid reconstruction from whole genome sequences**. *PLoS Genet* 2014, **10**(12):e1004766.

10. Shannon P, Markiel A, Ozier O, Baliga NS, Wang JT, Ramage D, Amin N, Schwikowski B, Ideker T: **Cytoscape: a software environment for integrated models of biomolecular interaction networks**. *Genome research* 2003, **13**(11):2498-2504.

11. Dayarian A, Michael TP, Sengupta AM: **SOPRA: Scaffolding algorithm for paired reads via statistical optimization**. *BMC bioinformatics* 2010, **11**:345.

12. Boetzer M, Henkel CV, Jansen HJ, Butler D, Pirovano W: **Scaffolding pre-assembled contigs using SSPACE**. *Bioinformatics (Oxford, England)* 2011, **27**(4):578-579.

13. Gordon D, Green P: **Consed: a graphical editor for next-generation sequencing**. *Bioinformatics* 2013, **29**(22):2936-2937.

14. Gordon D, Abajian C, Green P: **Consed: a graphical tool for sequence finishing**. *Genome Res* 1998, **8**(3):195-202.

15. Nielsen CB, Jackman SD, Birol I, Jones SJ: **ABySS-Explorer: visualizing genome sequence assemblies**. *IEEE transactions on visualization and computer graphics* 2009, **15**(6):881-888.

16. Riba-Grognuz O, Keller L, Falquet L, Xenarios I, Wurm Y: **Visualization and quality assessment of de novo genome assemblies**. *Bioinformatics (Oxford, England)* 2011, **27**(24):3425-3426.

17. Tang B, Wang Q, Yang M, Xie F, Zhu Y, Zhuo Y, Wang S, Gao H, Ding X, Zhang L *et al*: **ContigScape: a Cytoscape plugin facilitating microbial genome gap closing**. *BMC Genomics* 2013, **14**:289.

18. Wick RR, Schultz MB, Zobel J, Holt KE: **Bandage: interactive visualisation of *de novo* genome assemblies**. *bioRxiv* 2015.

19. Simpson JT, Wong K, Jackman SD, Schein JE, Jones SJ, Birol I: **ABySS: a parallel assembler for short read sequence data**. *Genome Res* 2009, **19**(6):1117-1123.

20. Zerbino DR, Birney E: **Velvet: algorithms for *de novo* short read assembly using de Bruijn graphs**. *Genome research* 2008, **18**(5):821-829.

21. Grabherr MG, Haas BJ, Yassour M, Levin JZ, Thompson DA, Amit I, Adiconis X, Fan L, Raychowdhury R, Zeng Q *et al*: **Full-length transcriptome assembly from RNA-Seq data without a reference genome**. *Nature biotechnology* 2011, **29**(7):644-652.

22. Camacho C, Coulouris G, Avagyan V, Ma N, Papadopoulos J, Bealer K, Madden TL: **BLAST+: architecture and applications**. *BMC bioinformatics* 2009, **10**:421.

23. Nakamura K, Oshima T, Morimoto T, Ikeda S, Yoshikawa H, Shiwa Y, Ishikawa S, Linak MC, Hirai A, Takahashi H *et al*: **Sequence-specific error profile of Illumina sequencers**. *Nucleic Acids Res* 2011, **39**(13):e90.

24. Crusoe MR, Edvenson G, Fish J, Howe A, McDonald E, Nahum J, Nanlohy K, Ortiz-Zuazaga H, Pell J, Simpson J *et al*: **The khmer software package: enabling efficient sequence analysis**: Figshare; 2014.

25. Langmead B, Salzberg SL: **Fast gapped-read alignment with Bowtie 2**. *Nat Meth* 2012, **9**(4):357-359.

26. McElroy K, Luciani F, Thomas T: **GemSIM: general, error-model based simulator of next-generation sequencing data**. *BMC Genomics* 2012, **13**(1):74.

27. Totsika M, Beatson SA, Sarkar S, Phan M-D, Petty NK, Bachmann N, Szubert M, Sidjabat HE, Paterson DL, Upton M *et al*: **Insights into a Multidrug Resistant *Escherichia coli* Pathogen of the Globally Disseminated ST131 Lineage: Genome Analysis and Virulence Mechanisms**. *PloS one* 2011, **6**(10):e26578.

28. Forde BM, Ben Zakour NL, Stanton-Cook M, Phan M-D, Totsika M, Peters KM, Chan KG, Schembri MA, Upton M, Beatson SA: **The Complete Genome Sequence of *Escherichia coli***

EC958: A High Quality Reference Sequence for the Globally Disseminated Multidrug Resistant *E. coli* O25b:H4-ST131 Clone. *PloS one* 2014, **9**(8):e104400.

29. Nguyen HA, Tomita T, Hirota M, Kaneko J, Hayashi T, Kamio Y: **DNA inversion in the tail fiber gene alters the host range specificity of carotovoricin Er, a phage-tail-like bacteriocin of phytopathogenic *Erwinia carotovora* subsp. carotovora Er**. *J Bacteriol* 2001, **183**(21):6274-6281.

30. Chen SL, Hung CS, Xu J, Reigstad CS, Magrini V, Sabo A, Blasiar D, Bieri T, Meyer RR, Ozersky P *et al*: **Identification of genes subject to positive selection in uropathogenic strains of *Escherichia coli*: a comparative genomics approach**. *Proc Natl Acad Sci U S A* 2006, **103**(15):5977-5982.

31. Nicholson TAJ: **Permutation procedure for minimising the number of crossings in a network**. *Electrical Engineers, Proceedings of the Institution of* 1968, **115**(1):21-26.