Bioinformatics, 34(15), 2018, 2556–2565
doi: 10.1093/bioinformatics/bty157
Advance Access Publication Date: 15 March 2018
Original Paper

Sequence analysis

Integrating long-range connectivity information into de Bruijn graphs

Isaac Turner[1,*,†], Kiran V. Garimella[1,2,*,†], Zamin Iqbal[1,3] and Gil McVean[1,2]

[1] Wellcome Trust Centre for Human Genetics, University of Oxford, Oxford, OX3 7BN, UK, [2]Big Data Institute, Li Ka Shing Centre for Health Information and Discovery, University of Oxford, Oxford, OX3 7LF, UK and [3]European Bioinformatics Institute (EMBL-EBI), Wellcome Genome Campus, Hinxton CB10 1SD, UK

*To whom correspondence should be addressed.
†The authors wish it to be known that, in their opinion, the first two authors should be regarded as Joint First Authors.
Associate Editor: Bonnie Berger

Received on June 8, 2017; revised on November 25, 2017; editorial decision on March 12, 2018; accepted on March 14, 2018

Abstract

Motivation: The de Bruijn graph is a simple and efficient data structure that is used in many areas of sequence analysis including genome assembly, read error correction and variant calling. The data structure has a single parameter *k*, is straightforward to implement and is tractable for large genomes with high sequencing depth. It also enables representation of multiple samples simultaneously to facilitate comparison. However, unlike the string graph, a de Bruijn graph does not retain long range information that is inherent in the read data. For this reason, applications that rely on de Bruijn graphs can produce sub-optimal results given their input data.
Results: We present a novel assembly graph data structure: the *Linked de Bruijn Graph (LdBG)*. Constructed by adding annotations on top of a de Bruijn graph, it stores long range connectivity information through the graph. We show that with error-free data it is possible to losslessly store and recover sequence from a Linked de Bruijn graph. With assembly simulations we demonstrate that the LdBG data structure outperforms both our de Bruijn graph and the String Graph Assembler (SGA). Finally we apply the LdBG to *Klebsiella pneumoniae* short read data to make large (12 kbp) variant calls, which we validate using PacBio sequencing data, and to characterize the genomic context of drug-resistance genes.
Availability and implementation: Linked de Bruijn Graphs and associated algorithms are implemented as part of McCortex, which is available under the MIT license at https://github.com/mcvean lab/mccortex.
Contact: kiran@well.ox.ac.uk or turner.isaac@gmail.com
Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

Most efforts to discover genetic variation in populations begin with alignment of high-throughput sequencing (HTS) data to a high-quality reference genome for the organism under study. This approach works well for regions with low divergence from the reference haplotype. However, many biologically interesting loci reside in regions of high divergence. For example, antigenic genes in *Plasmodium falciparum*, *Trypanosoma brucei* and other pathogens often exhibit non-allelic homologous recombination underlying mechanisms of immune escape (Artzy-Randrup *et al.*, 2012; Freitas-Junior *et al.*, 2000; Jackson *et al.*, 2012). Similarly, structural mutations, such as rearrangements and amplifications, can promote tumourigenesis through dysregulation of oncogenes or down-regulation of tumour suppressors (Aguilera and Gómez-González, 2008; Difilippantonio *et al.*, 2002). More generally, variants may be difficult to identify and characterize when the altered haplotype differs substantially from the reference, and other regions of interest reside in sequence absent from the reference sequence altogether. For example, in 13 isolates of the diploid coccolithophore *Emiliania huxleyi*, 8–40 Mbp of the approximately 142 Mbp genome were found to be isolate-specific; up to 25% of genes were found to be absent from the reference sequence (Read *et al.*, 2013). In these scenarios, reads may fail to map to the reference, preventing the analyst from inspecting biologically interesting variation. Alternatively, reads may map incorrectly, misleading the analyst to consider variation where none exists (Ribeiro *et al.*, 2015).

One mitigation of this inadequate reference problem is to augment the reference with known variation and alternative alleles to improve read mapping (Dilthey *et al.*, 2015; Huang *et al.*, 2013; Schneeberger *et al.*, 2009; Weisenfeld *et al.*, 2014). Such approaches commonly convert flat reference genomes into a graph structure, effectively mapping reads to all references simultaneously and choosing the path that best fits the data. In a study mapping to a fragmented human assembly, Limasset *et al.* (2016) found that mapping to a reference graph instead of flat contigs led to a 22% increase in the number of reads that map uniquely.

De novo assembly offers a means to overcome some of the limitations of reference-based analyses. Rather than aligning reads to a reference, reads are aligned to one another. These alignments are encoded in a graph data structure, a collection of 'vertices' encapsulating sequence data and 'edges' representing overlaps of different sequences (Myers, 1995). Graphs from different samples (and any reference) can then be compared to discover variation directly (Bateman *et al.*, 2016). Should the variation be in a locus unrepresented in the reference genome, the graph-based comparison can still capture the event (Iqbal *et al.*, 2012).

The most common sequencers in use today (second-generation) produce tens of millions of short reads (typically 75–150 bp in length) per sequencing run (Goodwin *et al.*, 2016). It is common to assemble such data using a so-called 'de Bruijn' graph approach (de Bruijn, 1946; Idury and Waterman, 1995; Pevzner, 1989). Vertices are constrained to be fixed-width substrings of length k (or 'k-mers'). Edges represent observed sequence adjacencies in the reads. With sufficient coverage, overlaps are implicitly encoded because two reads which overlap will share k-mers. Thus the graph is built up one read at a time at the cost of storing the graph in memory. Graphs of multiple individuals can be compared in memory (Iqbal *et al.*, 2012). However, there is a penalty for this approach: long-range information in the read is sacrificed. This is particularly problematic as genomes tend to have many repetitive regions and without context it is often not possible to determine the origin of a random k-mer (Miller *et al.*, 2010; Pevzner, 2004). However, as k increases, so does the specificity of its location. String graphs address the issue of storing long-range information by avoiding the read fragmentation step and instead find explicit overlaps between reads. Unfortunately string graphs are not well suited to multi-sample comparison and have a high per-sample memory cost (Bonizzoni *et al.*, 2016).

We start by describing the de Bruijn graph and its benefits compared to the string graph. We then describe an augmentation (LdBG) that allows long-range information to be kept. Theoretical results and simulations are used to characterize its properties. We demonstrate its value by application to variant discovery and characterization of genomic context for drug resistance genes in *Klebsiella pneumoniae*. Finally, we consider the possibility of using such structures for regular analysis of human-scale genomes.

2 Background

2.1 Definitions and notation

DNA sequences are strings over the alphabet {A,C,G,T}. We denote a DNA sequence as $S = S_1, \ldots, S_{|S|}$ where $|S|$ is the length of the sequence. $S[i,j]$ is sequence $S_i, \ldots, S_j$. S' is the reverse of S $(S_{|S|}, \ldots, S_1)$. $\bar{S}$ is the reverse complement of S. A k-mer is a sequence of length k over the alphabet {A,C,G,T}.

2.2 Assembly graphs

An assembly graph is any graph where the vertices represent sequence and edges represent overlaps or adjacencies between those sequences. An assembly graph may not have parallel edges (not a multigraph). Traversing a vertex v backwards $(\bar{v})$ gives the reverse complement of the sequence it represents. $deg^-(v)$ is the indegree and $deg^+(v)$ is the outdegree of vertex v. A path through the graph is a list of adjacent vertices with edges between them. The reverse of path $P = v_1, \ldots, v_n$ is $\bar{P} = \bar{v_n}, \ldots, \bar{v_1}$. A *unitig* $U = v_1, \ldots, v_n$, is a maximal path such that $deg^-(v_i) = deg^+(v_i) = 1$ for $1 < i < n$ and $deg^+(v_1) = deg^-(v_n) = 1$ if $n > 1$. The maximal property means the path cannot be extended without violating these conditions.

2.3 de Bruijn graphs

A de Bruijn graph $G(k)$ is an assembly graph, constructed from a set of sequence reads R and defined by $\{V, E\}$ where V is a set of vertices representing k-mers and E a set of edges between those k-mers. de Bruijn graphs are constructed by breaking input reads into overlapping k-mers that are added to the graph. With one k-mer starting at every base, a read of length $|r_i|$ will give $|r_i| - k + 1$ k-mers. A count is kept of how many times a given k-mer was seen in the input reads, called k-mer coverage. Edges are added between two k-mers if they share an overlap of k – 1 bases. Some implementations additionally require that k-mers are seen overlapping by k – 1 bases in the read data, in order to have an edge between them.

Due to the double stranded nature of DNA and the fact that we do not know which strand a read originated from, storing all k-mers from reads results in k-mers occurring separately in the graph in both their forward and reverse complement orientations. To overcome this it is common to store only the lexically lower of each k-mer X and its reverse-complement $\bar{X}$ (Zerbino, 2010). Requiring that k is odd prevents a k-mer from being its own reverse complement (a DNA palindrome). When visiting a vertex in the de Bruijn graph we can visit it in its forward or reverse complement orientation. The orientation in which we arrive at it determines if we leave by its out- or in- edges (forward, reverse respectively).

A de Bruijn graph only stores connectivity information one base either side of a given k-mer. This means that for three adjacent k-mers in the graph (v_a, v_b, v_c), there is no information about how the first and third are connected if the middle k-mer has $deg^-(v_b) > 1$ and $deg^+(v_b) > 1$. This graph motif is known as a 'tangle' and is caused by the graph collapsing down at a repeat and splitting out again afterwards. de Bruijn graphs collapse down at repeats in the genome of lengths $\geq k$. It is not possible to traverse a dBG past a tangle, even if the input reads are long enough to resolve it (i.e. pair-up k-mers going-into and coming-out of it). This makes analyses that use a de Bruijn graph sensitive to the parameter k.

While increasing k can overcome the problem of short repeats, it also has the effect of reducing the number of k-mers given by each read and increases the number of k-mers lost to each sequencing error. Both these effects reduce k-mer coverage, which is determined by the k-mer size, the read length and the error rate (Iqbal *et al.*, 2012). As k-mer coverage drops, read overlaps are lost and gaps in coverage increase. Together with tangles, coverage gaps interrupt assembly and shorten contigs.

Choosing a value for k is ultimately a trade-off. It is common to run analyses multiple times with different values of k and pick the best results according to a quality metric (e.g. assembly N50 or number of variants called) (Iqbal *et al.*, 2013). Alternatively the genome and read data can be sampled to estimate which value would be optimal (Zerbino, 2010).

The dBG can be augmented to support multiple datasets, providing a single data structure to describe and compare the genomes of many individuals (Iqbal *et al.*, 2012). Graphs are built separately for each dataset $c \in C$ and merged post-construction. The merge produces a union graph $G_u = \{V_u, E_u\}$, where $V_u = \cup_{c \in C} V_c$ and $E_u = \{E_c : c \in C\}$. Each k-mer stores which samples it was seen in. We refer to c as *colour*, a generic term that can mean a distinct individual, pooled population or a specific dataset on a single individual (e.g. tumour/normal), depending on analysis context. We shall refer to this structure as a *multi-colour de Bruijn graph*.

de Bruijn graphs are used in many areas of sequence analysis, including in mapping-based calling, as in the local alignment step of the variant caller Platypus (Rimmer *et al.*, 2014), in de novo assembly as in Velvet (Zerbino and Birney, 2008) and ABySS (Simpson *et al.*, 2009), and in de novo assembly for variant calling (Iqbal *et al.*, 2012).

Recently there has been work on implementing low memory dBG construction (Chikhi *et al.*, 2015), representations (Bowe *et al.*, 2012; Chikhi and Rizk, 2013; Conway and Bromage, 2011; Muggli *et al.*, 2017) and dBG-based compression (Benoit *et al.*, 2015; Holley *et al.*, 2017). These have both provided great improvements over the naive hash table based implementation, extending the contexts in which dBGs can be used.

2.4 String graphs

A String graph is an assembly graph where the vertices represent the input reads and the edges are maximal non-transitive overlaps between them (Myers, 2005). The set of reads is reduced to remove reads contained within other reads. A naïve String graph implementation would take $O(N^2)$ time to compare all pairs of reads to find overlaps, before removing contained reads and transitive edges. (Simpson and Durbin, 2010) showed that it is possible to construct a string graph in linear time, by first generating an FM-index (Ferragina and Manzini, 2000) of the input reads R and an FM-index of their reverse R'. Alternatively a single index can be constructed containing R and $\bar{R}$ (Li, 2012).

The FM-index is a data structure for compression and fast string searching. When the alphabet employed by the strings is small and constant-size (e.g. DNA nucleotides), the FM-index of a set of strings S facilitates searching for a query Q in time $O(|Q|)$ and has construction time and memory complexity $O(|S|)$. The final index has roughly the same size $|S|$, but can be efficiently compressed with run-length encoding.

Since it is constructed from the reads without breaking them up, a String graph retains all connectivity information contained in the single-ended input reads (Myers, 2005). However, String graphs do not naturally lend themselves to storing information on read pairs, although one such data structure has been proposed (Chikhi and Lavenier, 2011).

2.5 Other approaches for preserving connectivity

Reference-guided assembly can help overcome deficiencies in short-read dBG-based assembly by providing a template sequence to merge contigs that are likely adjacent but could not be merged due to repeats, coverage dropout, or error. For example, the Columbus module in Velvet allows for the specification of one or more reference genomes to aid in reconstruction, though documentation suggests that closely related references and repetitive regions may not work well with this software (see http://gensoft.pasteur.fr/docs/velvet/1.1.02/Columbus_manual.pdf). In contrast, RACA enables the use of a reference genome and multiple outgroups to guide the assembly of contigs and scaffolds from other assemblers (Kim *et al.*, 2013). However, RACA may introduce a reference bias as it computes synteny blocks by pairwise alignment solely against the single reference genome and omitting contigs that cannot be placed on the reference. Ragout addresses this concern by extending the syntenic block computation to multiple references, allowing for different block scales (Kolmogorov *et al.*, 2014).

Another strategy for preserving connectivity information in graphs is to reduce the dependence of dBGs on the single parameter k. One approach is to construct assemblies at multiple values of k, cluster resulting contigs across all assemblies by sequence similarity, choose a representative contig for each cluster (e.g. longest) and merge representative contigs by alignment to form a final assembly (A detailed procedure is specified at http://ged.msu.edu/angus/metag-assembly-2011/velvet-multik.html.). Alternatively, IDBA performs iterative assemblies starting at low values of k and removing reads utilized in the assembly before increasing k and repeating the process (Peng *et al.*, 2010). MEGAHIT employs a similar approach, removing likely sequencing errors in the graph rather than discarding reads after each iteration (Li *et al.*, 2015). These methods capture increasing amounts of connectivity information as k increases, but do not incorporate paired-end information. SPAdes builds on the iterative multi-k approach and its authors have explored using paired-k-mers (or '*k-bimer*') which encodes two k-mers and a distance estimate directly into the early stages of genome assembly (Bankevich *et al.*, 2012).

Beyond these multi-k approaches, various annotation schemes on de Bruijn graphs have emerged. The DARRC tool for read compression from (Holley *et al.*, 2017) uses a guided de Bruijn graph (gdBG), a multi-color de Bruijn graph constructed on sets of read sequences. While decomposition of reads into k-mers may produce false edges ($k-1$ overlaps present in the graph due to sequence similarity but not reflecting true genomic sequence), read information is stored as graph paths (effectively alignments) which disambiguate junction choices and provides a compressed representation of input data. Faucet, a two-pass streaming approach for *de novo* assembly graph construction, additionally records information from reads spanning adjacent junctions, used in the offline graph-simplification stage for disentangling repetitive graph regions (Rozov *et al.*, 2017). (Bolger *et al.*, 2017) have recently presented the LOGAN graph, which augments each vertex in a dBG with a routing table that pairs distant k-mers based on observation within the same read. This routing idea, adapted from pathfinding protocols in telecommunication networks, shares conceptual similarities to what we present below.

3 Materials and methods

3.1 The linked de Bruijn graph

We propose a new assembly graph data structure called the Linked de Bruijn Graph (LdBG). Defined as $LG(k) = (V, E, L)$ where V, E are defined as in a de Bruijn graph. $L(v)$ is a set of paths through the graph that start at vertex $v \in V$. We call these paths *links*. Each of these links $l \in L(v)$ is stored as a list of junction choices that when followed, starting from vertex v, recreate the path. Graph traversal is the same as with a de Bruijn graph, with the extension that when we visit a vertex v, we pick up the links associated with it: $L(v)$. The links held during traversal record how many edges ago they were picked up, a value we call link 'age'. Only when we reach a bifurcation in the graph do we consult the links currently held. We follow the next junction choice of the oldest link as this provides the most context as to where we are in the genome. Younger links are discarded if they are inconsistent with the oldest link. Should we have more than one oldest link and they disagree, we halt traversal. An illustration of links resolving a cycle is shown in Figure 1.

As with a de Bruijn graph we can look up any k-mer or edge between k-mers in time $O(1)$ and we can start graph traversal from any k-mer. As in a multi-coloured dBG, a multi-coloured LdBG stores which samples have which k-mers and links.

A LdBG is a lossless representation of a genome when generated from error-free reads, as long as the genome starts and ends with unique k-mers, there are no k-mer coverage gaps and each repeat is spanned by at least one read (proof in Supplementary Material). This is true regardless of the value of k.

In constructing a LdBG we are effectively compressing reads against the de Bruijn graph. However, since read start/end positions are not important for assembly we do not store them, so although it is possible to recover the underlying genome (losslessly) through assembly, it is not possible to recover the original set of input reads.

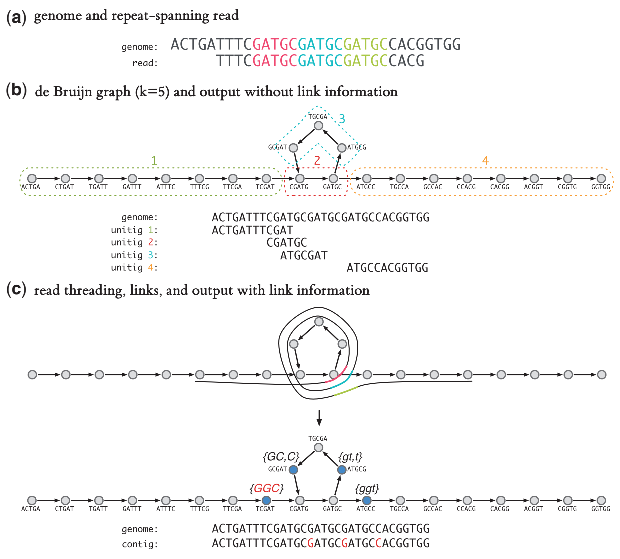
(a) genome and repeat-spanning read
genome: ACTGATTTCGATGCGATGCGATGCCACGGTGG
read: TTTCGATGCGATGCGATGCCACG
(b) de Bruijn graph (k=5) and output without link information
TGCGA
3
GCGAT
ATGCG
1
2
4
ACTGA
CTGAT
TGATT
GATTT
ATTTC
TTTCG
TTCGA
TCGAT
CGATG
GATGC
ATGCC
TGCCA
GCCAC
CCACG
CACGG
ACGGT
CGGTG
GGTGG
genome: ACTGATTTCGATGCGATGCGATGCCACGGTGG
unitig 1: ACTGATTTCGAT
unitig 2: CGATGC
unitig 3: ATGCGAT
unitig 4: ATGCCACGGTGG
(c) read threading, links, and output with link information
TGCGA
{GC,C}
GCGAT
{gt,t}
ATGCG
{GGC}
{ggt}
ACTGA
CTGAT
TGATT
GATTT
ATTTC
TTTCG
TTCGA
TCGAT
CGATG
GATGC
ATGCC
TGCCA
GCCAC
CCACG
CACGG
ACGGT
CGGTG
GGTGG
genome: ACTGATTTCGATGCGATGCGATGCCACGGTGG
contig: ACTGATTTCGATGCGATGCGATGCCACGGTGG

Fig. 1. Utility of link information in traversing a graph cycle. (**a**) A 32-bp genome and a 23-bp read, each containing three (colour-coded) repeats of the 5-mer, GATGC. (**b**) The resulting de Bruijn graph ($k=5$) with a repeat cycle, constructed from the genome sequence. The k-mers grouped by dashed boxes indicate the result of graph traversals to emit unitigs, with final sequences written below and positioned along the input genome for clarity. (**c**) Reads are 'threaded' (aligned) through the graph (top); the repeated k-mers are colour-coded. The alignment information is distilled to a set of junction choices to make when navigating the graph and stored as annotations on k-mers preceding junctions (middle). Multiple links are separated by a comma. Uppercase (lowercase) links indicate the choices to be made when traversing forwards (backwards). A k-mer's links are picked up when we visit it. When we reach a junction, the next edge suggested by the oldest link(s) is taken, links that disagree are dropped, all remaining links trim off a junction choice and exhausted links are also dropped. The resultant contig recapitulating the entire genome is shown (bottom). Highlighted bases indicate the junction choices originating from the left-most link

An LdBG on single-end reads can also be viewed as equivalent to a string graph with minimum overlap of $\tau_{min} = k$ bases and maximum error rate $\epsilon_{max} = 0$.

Reads used to annotate the graph do not need to have been used to construct the de Bruijn graph. Sets of links may be merged by loading them together at runtime. We give an example of the utility of such a construction in the applications section below.

3.2 de Bruijn graph construction

Each input read r is broken into $|r| - k + 1$ overlapping k-mers $(v_1, \ldots, v_n)$ which are added to the graph. If a k-mer already exists in the graph, we increment its coverage. Edges are added between v_i and v_{i+1} for all $1 \leq i < n$.

To remove k-mers due to sequencing error, unitigs with median k-mer coverage below T are removed, where T is a user-specified threshold. If not specified, a threshold T is chosen such that the expectation of a k-mer with coverage T being an error is $< 10^{-3}$ (see Supplementary Section S1 of Supplementary Material).

Graph tips, that is unitigs of length n (i.e. $v_1, \ldots, v_n$) with $deg^-(v_1) + deg^+(v_n) < 2$, are the result of sequencing errors near the end of reads or gaps in coverage. Tips are removed if they are shorter than a user specified value, the default being k, the maximum number of erroneous k-mers generated by a single-base sequencing error near the end of a read.

3.3 Read-to-graph alignment

Reads are aligned to the de Bruijn graph one-at-a-time and in doing so are error-corrected. For a read r, we look up each of its k-mers, resulting in a list of k-mers that describe a path through the graph. There may be gaps in this path due to k-mers removed from the graph during k-mer error cleaning (or if the read was not used in dBG construction). Gaps are closed by walking the graph between the k-mers either side of the gap (v_i and v_j). If we cannot traverse from v_i to v_j, we attempt going from v_j to v_i. Should such a traversal succeed (giving k-mers $v_1, \ldots, v_x$) and $x \sim |j - i|$, the k-mers $v_1, \ldots, v_x$ are used to fill in the gap in the path between v_i and v_j. This error step is sequence agnostic in that it does not compare the new k-mers $(v_1, \ldots, v_x)$ to the read k-mers it is replacing $(v_{i+1}, \ldots, v_{j-1})$. This speeds up the error correction step and ensures it does not make assumptions about the error process of the input sequence data. The output of the alignment step is a set of sequences that perfectly match the de Bruijn graph k-mers; i.e. they describe a path through the graph.

Gaps between paired-end reads are treated like gaps in reads caused by sequencing errors. LdBG naturally captures information from paired end reads once the insert gap is filled. Links can be generated in two passes: first with single-end reads against a dBG to create a LdBG; then with paired-end reads against the LdBG. This allows the single-ended read links to be used to aid traversal between read pairs.

3.4 Link annotation

A link is a path, starting from a given k-mer and stored as a series of junction choices. The function $J(v_i, \ldots, v_j)$ takes a path and returns the junction choices it describes.

Given a path $P = v_1, \ldots, v_n$ through the graph, we identify the maximum j such that $deg^+(v_j) > 1; 1 < j < n$. Then for each i such that $deg^-(v_i) > 1; 1 < i \leq j$, we add a link to vertex v_{i-1}: $L(v_{i-1}) \leftarrow L(v_{i-1}) \cup J(v_{i-1}, \ldots, v_{j+1})$. Link annotation is repeated for the reverse path $\bar{P}$. Link counts record how many times a given link is seen in a sample starting at a particular k-mer.

A collection of links (i.e. the series of junction choices arising from different reads spanning the same genomic locus) is easily represented as a tree, $L(v)$ (each node a junction, with possible outgoing edges A, C, G and T, with coverage incremented for each read supporting a junction choice). Cleaning links proceeds similarly to cleaning k-mers: junction choices with coverage below threshold T are discarded. This link cleaning threshold is determined by applying the same model as used for k-mer cleaning to the link coverage distribution of the first junction choice of all links (see Supplementary Section S1 of Supplementary Material).

3.5 Implementation

We have implemented the LdBG data structure and associated algorithms as part of McCortex, a modular set of multi-threaded programs for manipulating assembly graphs written in C. McCortex succeeds our cortex_var software for genome assembly and variant discovery, producing identical construction of multi-colour graphs and adding improvements for graph manipulation and link construction (The program cortex_con for consensus assembly, while co-developed alongside cortex_var, is not related to this work and is now retired.). FASTA, FASTQ, SAM, BAM & CRAM input file formats are supported. The software is released under the MIT license. McCortex has been used as the backend for sequence analysis by Bradley *et al.* (2015).

3.6 Multi-coloured linked de Bruijn graphs

Multi-colour LdBGs can be constructed by building single sample LdBGs and loading them together into McCortex. Links are represented internally as linked lists for which the only practical limit is machine memory. They can therefore be generated for any read length, including short Illumina reads (~100–200 bp), PacBio/MinION consensus sequences (~10 000–1 00 000 bp) and even whole chromosomes from finished reference sequences (potentially > 1e8 bp). For graph traversal tasks, such as assembly, we only store a single bit per sample per k-mer and per sample per link to record which k-mers/links are present in each sample. These are stored in a packed bitset. Graph traversal of a colour through a multi-coloured LdBG proceeds as per for a single-sample LdBG, only using links and k-mers of the given colour. At coverage gaps, traversal can fall back to using any k-mers in the graph (but not other colour's links).

4 Results: simulations

4.1 Equivalence of LdBG and input string

To test the lossless recovery of a genome from the LdBG we generated a random 10 kbp haploid genome, ensuring it started and ended with unique 7-mers. We identified the length of longest repeat (LR) in our genome. We generated perfect error-free coverage of the genome with a read length of $LR + 2$ starting at each base. We then built a LdBG ($k = 7$) from the reads, assembled contigs and removed contained contigs (those that were substrings of other contigs). After checking that we were left with a single contig, we compared it for an exact match to the original genome. This simulation was run 100 times without fail. With $k = 7$, there are only $4^7 = 16\,384$ possible k-mers, so a random 10 kbp genome will have many repeats that could not be traversed by the unannotated de Bruijn graph.

4.2 Correcting errors in reads

To assess the accuracy of our error correction step when aligning reads to the graph, we simulated a haploid 1 Mbp genome (from human GRCh37 chr22: 28 000 000–28 999 999). Single-ended 250 bp reads with 50× coverage were simulated with a 0.49% empirically distributed sequencing error (reads paired with real MiSeq data, FASTQ scores used as per base error rate). We built a dBG ($k = 31$) and removed tips and unitigs with coverage < 7 (automatically chosen). Once reads were aligned to the graph we wrote them to disk instead of generating links. The input reads had 247 075 (0.49%) errors, the output had 30 148 (0.06%) errors. Of the bases changed by the error correction step, 99.19% changes were correct.

4.3 Sensitivity to word length

Lowering the value of k in a dBG raises k-mer coverage and reduces coverage gaps but it also reduces the length of the longest repeats that can be traversed. If we improve the ability to resolve repeats with *links*, we hypothesized that we should reduce the assembly performance's sensitivity to the parameter k. Therefore we simulated an assembly task with different k values.

We simulated three haploid sequencing datasets from 1 Mbp of human (chr22: 28 000 000–28 999 999) using 100 bp single ended reads, each giving 100× coverage. First, we generated 'perfect coverage'—an error-free read starting at every base. Second, we generated 'stochastic coverage'—read starts distributed uniformly across the 1 Mbp genome. Third, we generated 'reads with error'—stochastically sampled reads with a uniform 0.5% rate of single base errors.

We assembled these three datasets using a dBG and LdBG at $k = 21, 31, \ldots, 91$. To compare assemblies we applied the QUAST tool for quality assessment of genome assemblies (Gurevich *et al.*, 2013) and examined the NG50 metric, defined as the contig length C such that contigs longer than C sum to at least half of the genome size. The NG50 comparisons are shown in Figure 2. In the *'perfect'* datasets reconstructed without links, NG50 rises as k-mer size increases. This is to be expected as a longer k-mer size essentially encodes more connectivity information. Links, however, encode all available connectivity information at any value of k. Thus the linked NG50 value (solid green line) is equal to the best unlinked NG50 (dashed green line) over all values of k. The *'stochastic'* datasets (orange) follow a similar pattern, with the exception that the top value of $k = 91$ does not necessarily yield better NG50. As read starts are

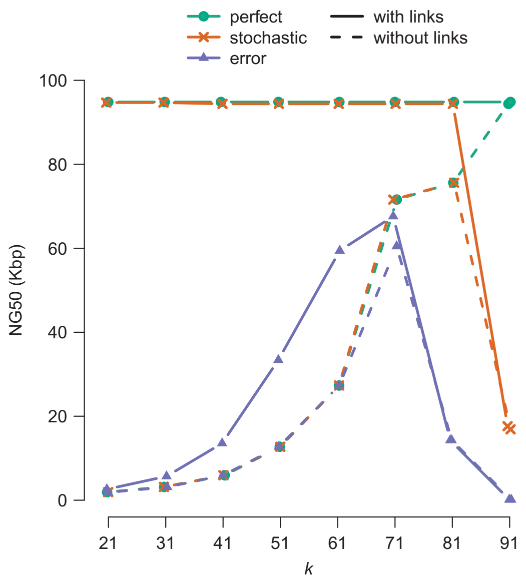

Fig. 2. Assembly length metric NG50 on raw de Bruijn graphs (i.e. without links, dashed lines) and linked de Bruijn graphs (i.e. with links, solid lines), as a function of k-mer size. Assembling 1 Mbp of sequence (human GRCh37 chr22: 28 000 000–28 999 999) with three simulated 100× read datasets: error free 100 bp reads, one read starting at every base (*'perfect'*, green); error free stochastic coverage (uniformly distributed read starts) (*'stochastic'*, orange); an error rate of 0.5% and stochastic coverage (*'error'*, purple)

not available at every single base, some read overlaps are not present and the resulting contig is thus truncated. Finally, the *'error'* dataset (blue) shows improved NG50 results when link information is used. When faced with sequencing error, our algorithms are not as readily capable of delivering k-independent reconstructions, although using links does improve performance at all values of k.

To explain this behaviour, we note that at low k, sequencing errors introduce false edges between true k-mers. Since error correction on dBGs use k-mer counts rather than edge counts, these false edges do not get cleaned off. We estimated the number of false edges induced at various k to be $604, 139, 30, 7, 2, 1, 0, 0, 0$ for $k = 21, 31, \ldots, 91$. Each false edge introduces a new bifurcation that may halt traversal. dBG implementations that use counts on edges instead of k-mers [as described in Conway and Bromage (2011)] may overcome this issue.

4.4 Comparison to other assemblers

McCortex is primarily intended for multi-sample comparison, rather than pure *de novo* assembly of organisms lacking reference sequence, and thus tends to be very conservative in its assembly procedure. However, its modular and flexible design does enable this usage, and thus we evaluated assembly performance between our LdBG implementation and other assemblers with different repeat resolution strategies. Velvet's 'Breadcrumb' module exploits read information to extend and connect contigs through repeats (Zerbino and Birney, 2008). The String Graph Assembler (SGA) (Simpson and Durbin, 2010) is able to use the full length of the read during assembly and should thus be able to assemble repeats shorter than a read length. SPAdes's ExSPAnder algorithm analyzes reads pairs that map to either side of an ambiguous junction, computing a confidence score based on the mapped versus expected insert sizes (Prjibelski *et al.*, 2014).

We compared all assemblers using the *E.coli* MG1655 dataset provided on the Illumina website (4.6 Mbp genome, 150 bp paired-end reads, 365 coverage). As evident in Figure 2, McCortex is sensitive to sequencing error. We applied the bfc error correction tool (Li, 2015) to the reads before assembling. Other assemblers have built-in error correction procedures, and bfc was therefore not applied. We did not carry out scaffolding for any tool. Detailed pipeline listings for the de Bruijn graph-based assemblers (McCortex, Velvet and SPAdes) and the string graph-based assembler (SGA) are provided in Supplementary Material Sections S3.2–S3.5.

All assemblers were run at values of k between 21 and 141 (except for SPAdes, which does not support k-mer sizes beyond 121) and compared using QUAST's NG50 metric. The results are shown in Figure 3. McCortex and SPAdes have fairly consistent performance across values of k; Velvet and SGA hit their optimums between $k = 91$ and sharp drop-offs before this value. With single-end reads alone, McCortex's performance is comparable to SGA, which is only capable of leveraging connectivity information inherent in single-end reads. We further improve performance through an iterative assembly procedure, e.g. by trimming 50 bp off both ends of contigs (as assembly errors will tend to appear at contig boundaries) and reassembling the results with the same procedure used for single-end reads. This iterative procedure enables McCortex to produce assemblies on par with SPAdes's performance.

5 Results: applications

To assess LdBG on real data, we examined short read data from *K.pneumoniae*, a gram-negative bacteria that usually lives harmlessly in the mouth and gut of humans. However in the event of a weakened immune system, it can establish pathogenic colonies in the lung leading to inflammation and bleeding. It is also found in some cases of urinary tract infections. Antibiotic resistant strains of *K.pneumoniae* have been found in patients. We used McCortex for two tasks where long-range information is likely to be beneficial—finding large differences from a reference and analysis of genomic context for drug resistance genes, which we validated using a PacBio reference assembled for the sample (Sheppard *et al.*, 2016).

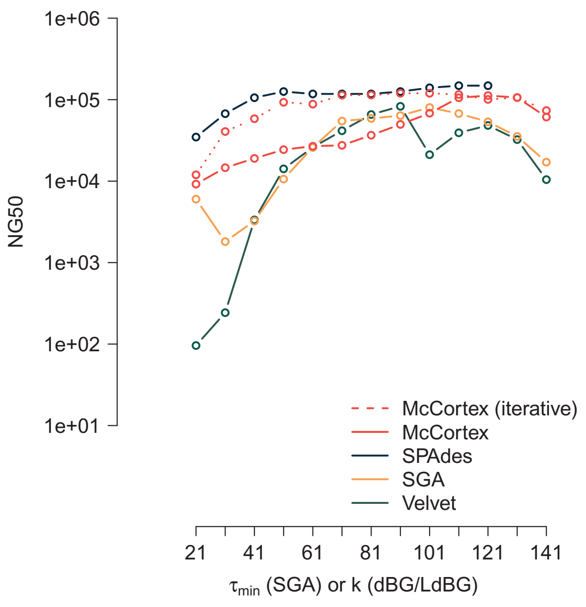

Fig. 3. Assembly NG50 results on an Illumina paired-end *E. coli* dataset (150 bp paired-end reads, 365× coverage) for McCortex, SPAdes, SGA, Velvet at various values of k (or τ for SGA). All assemblies are performed using single-end information only; paired-end data (threading or scaffolding) is not applied for any approach. For SPAdes, no results are available past $k = 121$, the software's maximum allowable setting. McCortex results are computed using single-end read threading, and an iterative scheme wherein the results from the single-ended assembly are further refined ('iterative')

5.1 Large-variant discovery

As links should provide useful guidance to navigating junctions in a graph, we examined their utility in calling large variants (insertions or deletions greater than 100 bp in length). We implemented a 'bubble caller' (named for the characteristic motif produced by a biallelic mutation in a graph wherein paths diverge from one k-mer and rejoin at another) and tested it by calling variants in CAV1016, a *K.pneumoniae* isolate for which a high-quality PacBio sequence was available for validation. We constructed dBGs of the canonical reference sequence (GCF_000016305.1_ASM1630v1) and Illumina data for CAV1016. From these, we built LdBGs using the single-end Illumina reads for link construction. We applied our bubble caller to the dBG and LdBGs, allowing for a minimum event size of 100 bp and maximum of 200 kbp, and removing duplicate events. We validated called alleles by aligning the reference and alternate alleles to the canonical reference sequence and CAV1016 PacBio sequence, respectively. The resulting callsets without and with link information are presented in Supplementary Table S1.

Using the dnadiff tool from the MUMmer package, we determined the canonical reference sequence and CAV1016 PacBio sequence had 35 141 SNPs and 779 indels (note however that as the dominant error mode of the PacBio instrument is indels, it is possible that many of the indels are false-positives). In the Illumina data, our bubble caller identified 27 706 SNPs and 550 indels, overlapping with the PacBio calls by 27 044 and 84 respectively. Examining the large events, 55 large indels were present in the dBG callset and 59 indels were in the LdBG callset. All 55 variants from the dBG callset were recovered in the LdBG callset. The four remaining variants exclusive to the LdBG callset are insertions of varying size (134; 246; 7952; and 11 946 bp).

That the LdBG-exclusive events should all be insertions (particularly large ones) is perhaps not surprising; in a graphical framework, calling insertions against a high-quality reference sequence with comparatively lower quality Illumina data is expected to be more difficult than calling deletions. With insertions, sequencing error in the study sample will produce spurious paths in the graph, not all of which can be removed successfully, and thus graph traversal from the 5′ to the 3′ end of the alternate allele has many opportunities to fail. With deletions, the graph navigation burden is on the reference allele which should have substantially fewer errors (and thus fewer spurious paths) to confound traversal. Link-informed traversal helps alleviate this insertion/deletion detection bias, enabling the recovery of large events like the 8 and 12 kb events listed above. This improves our access to large variants underrepresented in current variant call sets (Li, 2014b; Weisenfeld *et al.*, 2014).

5.2 Reference-link guided assembly

Finally, we show that with links, we can use a panel of reference contigs derived from multiple sources to improve drug resistance locus characterization in *K.pneumoniae* isolates. As the underlying graphs are considered immutable after construction, links derived from this panel cannot add *k*-mers to a sample. We hypothesized that the links panel could still provide valuable connectivity information where they were consistent with the graph without misleading the assembler in regions where they were divergent. We selected 21 *K.pneumoniae* isolates with known drug resistance status and that carry combinations of two alleles and two plasmid backgrounds at the *K.pneumoniae* carbapenemase (KPC) resistance locus, see Table 1. As references, we constructed links from a panel of four plasmid backgrounds carrying three different KPC alleles: PacBio sequences from two of the 21 isolates (carrying allele KPC-2), a KPC-harbouring plasmid from *E.coli* (carrying allele KPC-3) and a fourth *K.pneumoniae* plasmid known to harbour a resistance allele and background absent from the 21 isolates (carrying allele KPC-5). All accessions are described in Supplementary Table S2. Three assemblies were generated per isolate: one without links, one with links and one with the Columbus module in Velvet, using the links panel as input for reference-guided reconstruction.

Contigs harbouring the KPC sequence within the 21 isolates were identified by aligning to the KPC-2 allele sequence with LASTZ (Harris, 2007) and extracting the longest such contig from each assembly. These were aligned back to both the reference data sources and the validation data (Mathers *et al.*, 2015). For alignments that ran off the end of a sequence owing to the circular nature of the plasmids, we attempted to shift the contig sequence such that a linear alignment of maximum length was achieved; where this was not possible we have reported the length of the aligned region. The contig selected from each assembly was evaluated for correct KPC allele recovery, correct identification of plasmid background (i.e. sequence context of KPC allele), and mismatches/gaps to the relevant reference sequence. These results are shown in Table 1.

Without link information, we find that in 57% of cases the plasmid background on which the KPC allele resides cannot be identified. In such cases, LASTZ reports alignments of the short contigs with 100% sequence identity to plasmids 1 and 2. Moreover, for the CAV1360 isolate, the aligner determines the background incorrectly as the *E.coli* plasmid due to the presence of KPC-3.

Table 1. Comparison of KPC-containing contigs to validation data, inferred without/with links and with Velvet's Columbus module

		LdBG without link information			LdBG with link information			Velvet with Columbus		
Isolate	Known plasmid and KPC allele	Length (bp)	Matches uniquely and correctly	Mismatches, gaps	Length (bp)	Matches uniquely and correctly	Mismatches, gaps	Length (bp)	Matches uniquely and correctly	Mismatches, gaps
CAV1016	1, KPC-2	3893	Non-unique	0, 0	43 630	Yes	0, 0	34 240	Yes	7, 1
CAV1017	1, KPC-2	3895	Non-unique	0, 0	43 628	Yes	0, 0	10 001	Non-unique	0, 0
CAV1042	1, KPC-2	2557	Non-unique	0, 0	43 669	Yes	0, 0	9978	Non-unique	0, 0
CAV1077	1, KPC-2	3708	Non-unique	0, 1	15 796	Yes	0, 1	9879	Non-unique	0, 0
CAV1142	1, KPC-2	3892	Non-unique	0, 0	43 612	Yes	2, 0	9900	Non-unique	0, 0
CAV1145	1, KPC-2	5158	Yes	0, 0	43 643	Yes	1, 0	39 604	Yes	4, 2
CAV1182	1, KPC-2	2558	Non-unique	0, 0	9722	Yes	1, 0	9892	Non-unique	0, 0
CAV1203	1, KPC-2	3911	Non-unique	0, 0	43 670	Yes	0, 0	9993	Non-unique	0, 0
CAV1205	2, KPC-2	5978	Yes	0, 0	14 019	Yes	2, 1	52 582	Yes	24, 1
CAV1207	2, KPC-2	13 249	Yes	0, 0	13 991	Yes	3, 1	11 519	Yes	3, 0
CAV1237	1, KPC-2	20 444	Yes	1, 0	43 613	Yes	2, 1	34 177	Yes	7, 1
CAV1290	1, KPC-2	5099	Yes	0, 0	29 340	Yes	4, 1	9953	Yes	0, 0
CAV1292	1, KPC-2	5094	Yes	0, 0	29 337	Yes	1, 1	29 395	Yes	9, 3
CAV1338	2, KPC-2	5963	Yes	0, 0	14 003	Yes	2, 1	14 515	Yes	4, 0
CAV1344	1, KPC-2	4839	Yes	0, 0	9694	Yes	1, 0	9991	Non-unique	0, 0
CAV1351	1, KPC-3	3898	Incorrect	0, 0	43 616	Yes	1, 1	9953	Non-unique	0, 0
CAV1360	1, KPC-2	734	Non-unique	0, 0	43 621	Yes	1, 0	9922	Non-unique	0, 0
CAV1391	1, KPC-2	5106	Yes	0, 0	9726	Yes	1, 0	9918	Non-unique	0, 0
CAV1576	2, KPC-2	9161	Yes	0, 0	14 003	Yes	2, 1	10 105	Incorrect	0, 0
CAV1578	2, KPC-2	5243	Yes	0, 0	8956	Yes	0, 0	10 133	Incorrect	0, 0
CAV1597	2, KPC-2	12 627	Yes	0, 0	14 009	Yes	2, 1	14 037	Yes	19, 4

Reconstruction with the link panel provides an order of magnitude increase in contig length over the link-uninformed reconstructions and the inferred plasmid membership matches the Mathers *et al.* (2015) determination in all 21 cases. In contrast, the Velvet reconstructions are longer than the link-uninformed contigs, but only resolve the plasmid background correctly in two additional cases. It fails to determine a unique membership in two cases, and establishes an incorrect background in two other cases. The much higher mismatch and gap rate of the Velvet contigs suggest that these assemblies are more errorful. This is perhaps expected, as the Columbus module is designed for guided assembly with a single reference, rather than a panel of references. With considerable homology between the supplied sequences, the initial step of mapping reads to the reference panel is likely to produce erroneous alignments, confounding Velvet's ability to leverage the panel properly. We attempted to overcome this limitation by using Ragout to refine the Velvet and link-uninformed assemblies with multiple references. This effort failed as Ragout would not accept panel members having limited synteny with the assembled contigs.

Reconstructions from two isolates stand out. CAV1351 was known to carry the KPC-3 allele, while all other isolates carried the KPC-2 allele. The link-uninformed assembly produces a contig that maps to the *E.coli* KPC-3 sequence perfectly, but infers the wrong plasmid membership. The link-informed reconstruction, however, produces both the correct plasmid membership and correct allele. In another case, Mathers *et al.* (2015) reported CAV1077 to possess plasmid 1, but with an unspecified sequence alteration. Our reconstruction is able to establish both the correct plasmid membership and identify a 188 bp deletion in the intergenic region upstream from the transposase and downstream from the KPC genes (detected with and without links, but not with Velvet). Combined, these analyses demonstrate how using external data sources as a means to guide assembly through 'reference links' can lead to highly accurate reconstruction of even complex regions of the genome.

5.3 Scalability

Finally, to assess the scalability of LdBG to larger genomes, we constructed LdBGs ($k = 47$) from paired-end data for genomes ranging from ~5 Mbp (*E.coli*) to 3200 Mbp (*H.sapiens*). The full results are presented in Supplementary Table S3, using SGA's performance on the same datasets as a point of comparison (with the exception of the human dataset, due to high computational burden). In all cases, McCortex memory usage substantially exceeds that of SGA, while concomitant runtime is lower. The design choices between the two tools are clearly evident in the runtime results, with SGA explicitly optimized to reduce memory usage at the expense of CPU time, and McCortex opting to keep graphs and link information in memory to reduce CPU time. On the largest dataset tested, paired-end data from human sample NA12878, construction had a peak memory usage of 400 GiB (required to store the entire graph, including sequencing errors). After error cleaning, loading the dataset requires 70 GiB of RAM (50 GiB for the dBG, 20 GiB for links). Link construction is easily parallelized (the alignment of reads to the graph has no dependency on other reads). Using 32 threads for single- and paired-end link construction, a human genome covered to 25× is completed in under 72 h.

6 Discussion

We have presented a de novo assembly method that addresses the most important limitation of de Bruijn graphs: the ability to leverage long-range connectivity information inherent in read data. While cutting reads into small *k*-mers has long been a useful way of simultaneously computing read-to-read overlaps and overcoming high rates of sequencing error, increasing sequencing quality and read lengths have rendered de Bruijn assembly methods less attractive. String graphs have been successful in incorporating long-range data into assemblies, but sacrifice desirable computational properties of de Bruijn graphs. Our solution, Linked de Bruijn Graphs, combine the connectivity properties of string graphs with the rapid lookup of specific (multi-coloured) *k*-mers. Due to the wide range of uses of dBGs in sequence analysis, we believe this offers a potential improvement to many existing algorithms. Path encoding of reads has been suggested for read compression before (Conway and Bromage, 2011; Kingsford and Patro, 2015). However we believe this is the first implementation to use it for multi-colour assembly that can scale up to large mammalian genomes on modern computer hardware.

We have shown that read error correction and graph annotation can improve assembly performance of de Bruijn graphs and that this can be seen with the recovery of large (12 kbp) events in short read sequences. Moreover, through application to real data we have shown that links can be generated from a wide range of sequencing technologies including data not used to construct the underlying dBG, and that this can be exploited to identify sequences of biological interest. LdBGs can also naturally represent paired-end connectivity (where we treat filling in sequence between paired-end reads as the equivalent problem to closing gaps in alignment) and chimeric reads (e.g. chimeric reads spanning a translocation breakpoint would yield *k*-mers not necessarily present in a reference sequence, and alignment of these reads back to the graph would provide connectivity across the breakpoint) information. We have proven that in the error-free setting, Linked de Bruijn Graphs losslessly store the genome sequence, even when constructed from short reads and agnostic of *k*.

Our method is useful for reconstructing complex loci across multiple samples using a common panel of pre-determined haplotypes. Link information derived from a haplotype panel cannot add *k*-mers or edges to the graph that were not observed in the original dataset. Nevertheless, assembly is enhanced in regions where the links are consistent with the graph, and naturally defaults to link-uninformed navigation in regions of discrepancy. Threading a panel of haplotypes from multiple samples through each graph thus identifies only the relevant sections of each donor haplotype.

One shortcoming of long links is the accumulated probability of encountering an error during traversal. If a link takes the wrong branch of an error-induced bubble, cleaning that junction choice trims off all the remaining information about the junction choices made beyond the bubble. This shortcoming results in link coverage dropping off quicker than expected as links get longer, resulting in truncated links. This could be addressed by error-correcting groups of links that start at the same *k*-mer.

We have implemented a very simple read mapping, which trusts all *k*-mers from a read if there is a perfect match in the graph, and which only attempts to fill gaps using linear time graph traversal. Optimal mapping is ultimately NP-hard, but more advanced heuristic methods are available which may perform better than our approach (Limasset *et al.*, 2016). Improvement may be most noticeable for high error rate sequencing data and in low complexity regions of the graph.

More sophisticated graphical alignment approaches may also enable the use of uncorrected (i.e. low accuracy) reads from third generation sequencing for alignment, graph construction, or both. High error rate reads pose a problem in our current framework as the alignment is predicated on exact matches between a number of read and graph *k*-mers, under the assumption that the *k*-mers in the graph are correct. Even if the graph is built with a small *k*-mer size (e.g. 31 bp), uncorrected long reads may not contain enough perfect *k*-mers for alignment. Reducing the *k*-mer size of the graph may not solve this problem, as the number of false edges arising from homology would be expected to increase substantially, impeding our ability to traverse the graph to fill in error-induced gaps. Finally, construction of both the graph and links from uncorrected long reads is likely not possible using our method. Our error cleaning steps are not suited to this data type, and a naïve application of our software might simply ratify the errors inherent in the read data. We leave the pursuit of a LdBG implementation appropriate to uncorrected long read datasets to future work.

There is scope for reducing memory consumption, given very few *k*-mers actually have links attached (see Supplementary Fig. S6) and could be further reduced with better encoding in memory of the junction choice tree held by a *k*-mer (i.e. $L(v)$). For example, using a binary encoding of the tree of junction choices, or generative path encoding proposed to compress sequence data (Kingsford and Patro, 2015).

Acknowledgements

We thank Jerome Kelleher for his many useful comments and edits, Rachel Norris for her pointers and insight into the *K.pneumoniae* dataset, and the other members of the McVean lab for useful discussions during the preparation of this manuscript. We are grateful to our Bioinformatics reviewers (Rayan Chikhi and Erik Garrison) and BioRxiv preprint reviewers (Camille Scott, Titus Brown and Pall Melsted).

Funding

This work was supported by the Wellcome Trust (grant numbers 090532/Z/09/Z and 100956/Z/13/Z). K.V.G. was supported by Wellcome Trust Research Studentship award (097310/Z/11/Z). I.T. was supported by a PhD studentship from the BBSRC. Z.I. was funded by a Wellcome Trust/Royal Society Sir Henry Dale Fellowship (grant 102541/Z/13/Z).

Conflict of Interest: none declared.

References

Aguilera,A. and Gómez-González,B. (2008) Genome instability: a mechanistic view of its causes and consequences. *Nat. Rev. Genet.*, **9**, 204–217.

Artzy-Randrup,Y. *et al.* (2012) Population structuring of multi-copy, antigen-encoding genes in *Plasmodium falciparum*. *eLife*, **1**, e00093.

Bankevich,A. *et al.* (2012) SPAdes: a new genome assembly algorithm and its applications to single-cell sequencing. *J. Comput. Biol. J. Comput. Mol. Cell Biol.*, **19**, 455–477.

Bateman,A. *et al.* (2016) Limitations of current approaches for reference-free, graph-based variant detection. In: *Proceedings of the 7th ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics*, New York, NY, USA. ACM, pp. 499–500.

Benoit,G. *et al.* (2015) Reference-free compression of high throughput sequencing data with a probabilistic de Bruijn graph. *BMC Bioinformatics*, **16**, 288.

Bolger,A.M. *et al.* (2017). LOGAN: a framework for LOssless Graph-based ANalysis of high throughput sequence data. *bioRxiv*, p. 175976.

Bonizzoni,P. *et al.* (2016) An external-memory algorithm for string graph construction. *Algorithmica*, **78**, 394–424.

Bowe,A. *et al.* (2012) Succinct de Bruijn Graphs. In: Raphael,B. and Tang,J. (eds.) *Algorithms in Bioinformatics*. Springer, Berlin, pp. 225–235.

Bradley,P. *et al.* (2015) Rapid antibiotic-resistance predictions from genome sequence data for *Staphylococcus aureus* and *Mycobacterium tuberculosis*. *Nat. Commun.*, **6**, 10063–10063.

Bradnam,K.R. *et al.* (2013) Assemblathon 2: evaluating de novo methods of genome assembly in three vertebrate species. *GigaScience*, **2**, 10–10.

Chikhi,R. and Lavenier,D. (2011). Localized genome assembly from reads to scaffolds: practical traversal of the paired string graph. In: Przytycka,T. and Sagot,M.-F. (eds.) *Algorithms in Bioinformatics*. Springer, Berlin, pp. 39–48.

Chikhi,R. and Rizk,G. (2013) Space-efficient and exact de Bruijn graph representation based on a Bloom filter. *Algorithms Mol. Biol.*, **8**, 22.

Chikhi,R. *et al.* (2015) On the representation of de Bruijn graphs. *J. Comput. Biol.*, **22**, 336–352.

Conway,T.C. and Bromage,A.J. (2011) Succinct data structures for assembling large genomes. *Bioinformatics*, **27**, 479–486.

de Bruijn,N.G. (1946) A Combinatorial Problem. *Koninklijke Nederlandsche Akademie Van Wetenschappen*, **49**, 758–764.

Difilippantonio,M.J. *et al.* (2002) Evidence for replicative repair of DNA double-strand breaks leading to oncogenic translocation and gene amplification. *J. Exp. Med.*, **196**, 469–480.

Dilthey,A. *et al.* (2015) Improved genome inference in the MHC using a population reference graph. *Nat. Genet.*, **47**, 682–688.

Ferragina,P. and Manzini,G. (2000). Opportunistic data structures with applications. In: *Proceedings. 41st Annual Symposium on Foundations of Computer Science, 2000*, Los Alamitos, CA, USA. IEEE, pp. 390–398.

Freitas-Junior,L.H. *et al.* (2000) Frequent ectopic recombination of virulence factor genes in telomeric chromosome clusters of *P. falciparum*. *Nature*, **407**, 1018–1022.

Goodwin,S. *et al.* (2016) Coming of age: ten years of next-generation sequencing technologies. *Nat. Rev. Genet.*, **17**, 333–351.

Gurevich,A. *et al.* (2013) QUAST: quality assessment tool for genome assemblies. *Bioinformatics (Oxford, England)*, **29**, 1072–1075.

Harris,R.S. (2007) *Improved Pairwise Alignment of Genomic DNA*. PhD thesis, Pennsylvania State Univ.

Holley,G. *et al.* (2017) Dynamic alignment-free and reference-free read compression. In: *Research in Computational Molecular Biology*. Springer, Cham, pp. 50–65.

Huang,L. *et al.* (2013) Short read alignment with populations of genomes. *Bioinformatics*, **29**, i361–i370.

Idury,R.M. and Waterman,M.S. (1995) A new algorithm for DNA sequence assembly. *J. Comput. Biol.*, **2**, 291–306.

Iqbal,Z. *et al.* (2012) De novo assembly and genotyping of variants using colored de Bruijn graphs. *Nat. Genet.*, **44**, 226–232.

Iqbal,Z. *et al.* (2013) High-throughput microbial population genomics using the Cortex variation assembler. *Bioinformatics*, **29**, 275–276.

Jackson,A.P. *et al.* (2012) Antigenic diversity is generated by distinct evolutionary mechanisms in African trypanosome species. *Proc. Natl. Acad. Sci. USA*, **109**, 3416–3421.

Kim,J. *et al.* (2013) Reference-assisted chromosome assembly. *Proc. Natl. Acad. Sci. USA*, **110**, 1785–1790.

Kingsford,C. and Patro,R. (2015) Reference-based compression of short-read sequences using path encoding. *Bioinformatics*, **31**, 1920–1928.

Li,D. *et al.* (2015) MEGAHIT: an ultra-fast single-node solution for large and complex metagenomics assembly via succinct de Bruijn graph. *Bioinformatics (Oxford, England)*, **31**, 1674–1676.

Li,H. (2012) Exploring single-sample SNP and INDEL calling with whole-genome de novo assembly. *Bioinformatics*, **28**, 1838–1844.

Li,H. (2014a) Fast construction of FM-index for long sequence reads. *Bioinformatics*, **30**, 3274–3275.

Li,H. (2014b) Toward better understanding of artifacts in variant calling from high-coverage samples. *Bioinformatics*, **30**, 2843–2851.

Li,H. (2015) BFC: correcting Illumina sequencing errors. *Bioinformatics*, **31**, 2885–2887.

Limasset,A. *et al.* (2016) Read mapping on de Bruijn graphs. *BMC Bioinformatics*, **17**, 237–237.

Kolmogorov,M. *et al.* (2014) Ragout—a reference-assisted assembly tool for bacterial genomes. *Bioinformatics*, **30**, i302–i309.

Mathers,A.J. *et al.* (2015) *Klebsiella pneumoniae* carbapenemase (KPC) producing *K. pneumoniae* at a Single Institution: insights into Endemicity from Whole Genome Sequencing. *Antimicrob. Agents Chemother.*, **59**, 1656–1663.

Miller,J.R. *et al.* (2010) Assembly algorithms for next-generation sequencing data. *Genomics*, **95**, 315–327.

Muggli,M.D. *et al.* (2017) Succinct colored de Bruijn graphs. *Bioinformatics*, **33**, 3181–3187.

Myers,E.W. (1995) Toward simplifying and accurately formulating fragment assembly. *J. Comput. Biol.*, **2**, 275–290.

Myers,E.W. (2005) The fragment assembly string graph. *Bioinformatics*, **21**, ii79–ii85.

Peng,Y. *et al.* (2010) IDBA – a practical iterative de Bruijn graph de novo assembler. *RECOMB*, **6044**, 426.

Pevzner,P.A. (1989) l-Tuple DNA sequencing: computer analysis. *J. Biomol. Struct. Dyn.*, **7**, 63–73.

Pevzner,P.A. (2004) De novo repeat classification and fragment assembly. *Genome Res.*, **14**, 1786–1796.

Prjibelski,A.D. *et al.* (2014) ExSPAnder: a universal repeat resolver for DNA fragment assembly. *Bioinformatics*, **30**, i293–i301.

Read,B.A. *et al.* (2013) Pan genome of the phytoplankton *Emiliania underpins* its global distribution. *Nature*, **499**, 209–213.

Ribeiro,A. *et al.* (2015) An investigation of causes of false positive single nucleotide polymorphisms using simulated reads from a small eukaryote genome. *BMC Bioinformatics*, **16**, 382.

Rimmer,A. *et al.* (2014) Integrating mapping-, assembly- and haplotype-based approaches for calling variants in clinical sequencing applications. *Nat. Genet.*, **46**, 912–918.

Rozov,R. *et al.* (2017) Faucet: streaming de novo assembly graph construction. bioRxiv, **34**, 147–154.

Schneeberger,K. *et al.* (2009) Simultaneous alignment of short reads against multiple genomes. *Genome Biol.*, **10**, R98.

Sheppard,A.E. *et al.* (2016) Nested Russian doll-like genetic mobility drives rapid dissemination of the carbapenem resistance gene blaKPC. *Antimicrob. Agents Chemother.*, **60**, 3767–3778.

Simpson,J.T. and Durbin,R. (2010) Efficient construction of an assembly string graph using the FM-index. *Bioinformatics*, **26**, i367–i373.

Simpson,J.T. *et al.* (2009) ABySS: a parallel assembler for short read sequence data. *Genome Res.*, **19**, 1117–1123.

Weisenfeld,N.I. *et al.* (2014) Comprehensive variation discovery in single human genomes. *Nat. Genet.*, **46**, 1350–1355.

Zerbino,D.R. (2010) Using the Velvet de novo assembler for short-read sequencing technologies. *Curr. Protoc. Bioinf.*, **31**, 11.5.1–11.5.12.

Zerbino,D.R. and Birney,E. (2008) Velvet: algorithms for de novo short read assembly using de Bruijn graphs. *Genome Res.*, **18**, 821–829.