Konnector: Connecting Paired-end Reads Using a Bloom Filter de Bruijn Graph

Benjamin P Vandervalk, Shaun D Jackman, Anthony Raymond, Hamid Mohamadi, Chen Yang, Dean A Attali, Justin Chu, René L Warren, Inanç Birol

Genome Sciences Centre
BC Cancer Agency
Vancouver, Canada
e-mail: ibirol@bcgsc.ca

***Abstract*—Paired-end sequencing yields a read from each end of a DNA fragment, typically leaving a gap of unsequenced nucleotides in the middle. Closing this gap using information from other reads in the same sequencing experiment offers the potential to generate longer "pseudo-reads" using short read sequencing platforms. Such long reads may benefit downstream applications such as *de novo* sequence assembly, gap filling, and variant detection. With these possible applications in mind, we have developed Konnector, a software tool to fill in the nucleotides of the sequence gap between read pairs by navigating a de Bruijn graph. Konnector represents the de Bruijn graph using a Bloom filter, a probabilistic and memory-efficient data structure. Our implementation is able to store the de Bruijn graph using a mean 1.5 bytes of memory per k-mer, which represents a marked improvement over the typical hash table data structure. The memory usage per k-mer is independent of the k-mer length, enabling application of the tool to large genomes. We report the performance of the tool on simulated and experimental datasets, and discuss its utility for downstream analysis.**

***Availability*—Konnector is open-source software, free for academic use, released under the British Columbia Cancer Agency's academic license. The tool is included with ABySS version 1.5.2 and later, and is available for download from http://www.bcgsc.ca/platform/bioinfo/software/abyss.**

Bloom filter; de Bruijn graph; paired-end sequencing; de novo genome assembly

I. INTRODUCTION

A key characteristic of both first-generation (e.g. Sanger) and current second-generation sequencing methods (e.g. Illumina HiSeq) is the use of a paired-end tag (PET) sequencing strategy. Under a PET sequencing approach, short DNA sequences (typically 75-300 bp) are determined from both ends of a DNA fragment, and the pairing relationship of the sequences is recorded for downstream use. Although the size of the sequenced DNA fragments is variable, typically a unimodal distribution that may be modeled as a Gaussian, the pairing relationship between the reads provides valuable distance information that has been successfully exploited in several applications.

For example, many assembly pipelines incorporate *scaffolding algorithms* that map paired-end reads across contigs to determine their relative distances and ordering [1-4]. In addition, many stand-alone scaffolding algorithms have been implemented outside of assembly packages that use this pairing information [5-7]. Another application of paired-end reads is in detection of structural variations and mutations, where observed fragment lengths and orientations are compared with expected fragment length distributions and orientations to infer aberrant cases [8-11].

As read lengths from second-generation sequencing platforms increased, it has become possible to extend the paired-end reads far enough into the DNA fragments to allow them to overlap. This overlap presents an opportunity to combine the reads into a single long sequence, which has alternately been referred to as a "pseudo-read", "merged read", or "super-read". Numerous tools have been developed to merge overlapping paired-end reads [12, 13], and their evaluation has demonstrated that using pseudo-reads as input to assembly pipelines significantly improves the assembly quality.

When the read pairs either do not overlap at all or do not have a long enough overlap to be deemed specific enough, the gaps may still be filled using the sampling redundancy in the sequencing data. In the ELOPER algorithm [14], this is achieved by first identifying gapped overlaps between read pairs, and then merging them, to generate "elongated paired-end reads". The GapFiller algorithm [15], on the other hand, formulates this problem as a collection of seed-and-extend local assembly problems. The latter concept has also been implemented within the MaSuRCA *de novo* assembly algorithm [16].

One important concern for these pseudo-read technologies is their computational efficiency. The sequence throughput of established NGS platforms such as Illumina's has already increased by three orders of magnitude since their inception, reaching a terabase per run. Over the last few years, genomics research has benefited from advances in computer science, and clever strategies that make use of parallel computing [1, 17], FM indexing [18, 19], and compressed data structures [4], among others, have been proposed for handling big data.

978-1-4799-5669-2/14/$31.00 ©2014 IEEE

In this work, we introduce a new scalable algorithm to connect read pairs and its implementation, Konnector. The tool makes use of the redundancy in sequence coverage to fill the sequence gap between read pairs, and is capable of handling large sequence datasets from Gbp scale genomes by representing a de Bruijn graph for sequence overlaps in a Bloom filter, a memory-efficient data structure. We benchmark Konnector on simulated datasets, compare its performance against recent pseudo-read generating tools, and discuss several possible applications for the technology. We expect Konnector to have broad utility in genomics.

II. ALGORITHM

Konnector operates in three steps. In the first step, the k-mers from all paired-end reads are loaded into a Bloom filter implicitly representing a de Bruijn graph. In the second step, a bidirectional graph search is carried out between each pair of reads to find connecting paths. In the third and final step, a pseudoread is generated for each connected pair, which includes both the original read sequences and the sequence of the intervening gap. In cases where there are multiple paths joining a read pair, a consensus sequence is constructed from the alternate paths.

A. The Bloom Filter de Bruijn Graph Data Structure

A Bloom filter [20, 21] is a probabilistic and memory-efficient data structure that represents a set of elements. A de Bruijn graph may be represented using a Bloom filter, where the presence or absence of a k-mer (a subsequence of length k bp) in the graph is indicated by the Bloom filter. The use of a Bloom filter to represent a de Bruijn graph has been described previously for partitioning assembly graphs [22] and genome sequence assembly [23]. The latter application requires a second non-probabilistic data structure to enumerate the critical false positives. We note that this additional data structure is unnecessary for connecting reads, thereby reducing memory requirements through the use of a cascading Bloom filter, as described below.

We note that k-mers that are observed only once in the data are usually erroneous, and may thus be discarded. Also of note, a Bloom filter can only indicate the presence or absence of a k-mer, but cannot count the number of occurrences, due to its characteristic representation of only one bit per set element. A variation of the Bloom filter, called a counting Bloom filter, uses multiple bits per element, and can therefore be used to return not only presence or absence of a k-mer, but also a lower bound on the number of its occurrences. For example, using two bits per element, a counting Bloom filter can report if a k-mer occurs 0, 1, 2 or 3+ times.

In Konnector, rather than using a counting filter, we use a cascade of two Bloom filters. When inserting a k-mer, we check for its presence in the first filter. If it is not found, it is inserted into the first filter. If the k-mer is found in the first filter, it is also inserted into the second filter.

Using a cascade of two equal-sized Bloom filters doubles the memory requirement in our implementation to 10 bits per k-mer. One advantage of using cascade Bloom filters is that, once they are constructed, the first filter may be discarded, reducing the memory requirement back to 5 bits per k-mer. Although both Bloom filters are the same size in our implementation, the second filter could be smaller since fewer k-mer are inserted, further reducing memory usage.

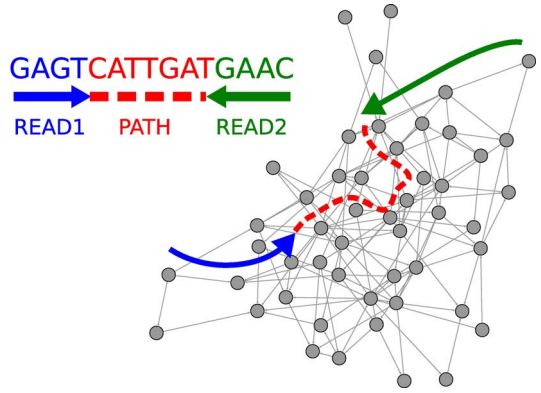

Figure 1. Konnector performs a bidirectional search between paired-end reads in the de Bruijn graph.

If a k-mer is present in the reads (more than once, in our implementation), the Bloom filter will certainly return true. However, if a k-mer is not present, the Bloom filter may incorrectly return true with some probability termed the false positive rate (FPR). The FPR is determined by the size of the Bloom filter, the number of distinct elements inserted and the number of hash functions.

Using percolation theory, the khmer paper [22] showed that an appropriate FPR for a DNA de Bruijn graph is 0.183, and that the optimal number of hash functions for this FPR is two. Using additional hash functions may lower the FPR, but also comes with a computational cost. To improve performance and for ease of implementation, we use a single hash function, which comes at the cost of an increase in memory usage of 40% from the theoretical optimum of 3.6 bits per k-mer to 5.0 bits per k-mer. Even at this sub-optimal parameterization, we note that Konnector is scalable for large-scale problems. For example, the de Bruijn graph of the 20-gigabase white spruce genome sequencing data [24] can be represented in 40 gigabytes.

B. Finding Connecting Paths within the de Bruijn Graph

The Bloom filter stores only the set of k-mers that are present in the de Bruijn graph; information about edges is not recorded. Instead, Konnector traverses edges by querying the Bloom filter for each of the four possible neighboring k-mers (1 bp extensions of A, C, G or T) at every step.

The search for paths between paired-end reads is subject to several constraints. The k-mers that make up the first and second reads constitute a set of possible start and end nodes for the path search, and any path that connects k-mers in the two sets is a potential solution. It is important to identify all possible paths between the first and second reads in order to ensure the correctness of any DNA sequences that are generated by the tool. The sequences produced from a unique connecting path are highly likely to be correct (with the exception of systematic sequencing errors), whereas merging read pairs with alternate connecting paths may or

may not be desirable, depending on the similarity of the paths. Finally, the connecting paths should be constrained to a minimum and maximum length as dictated by the DNA fragment size distribution that produced the reads.

The first step of the search algorithm is to select a single *start k-mer* from the first read and a single *goal k-mer* from the second read as endpoints for the path search. The choice of start and goal k-mers can have a significant effect on the speed of the algorithm because it alters the distance between the path endpoints and hence the maximum depth limit of the search. For this reason, the default behavior of the algorithm is to select start and goal k-mers that are as close as possible to the ends of the reads. Because it is not guaranteed that every k-mer in a read will be present in the graph (as discussed in the previous subsection) we assume that these start and goal k-mers are present in the Bloom filter, and walk back from read ends until this condition is satisfied.

The graph search algorithm used by Konnector is a depth-limited, bidirectional breadth-first search. Bidirectional search algorithms reduce the expansion of the search frontier, and thus offer significant performance advantages over unidirectional search algorithms. Breadth-first search was chosen over depth-first-search for this application because it allows the minimum distance of each node from the start/goal k-mer to be tracked during the traversal, which is in turn used to enforce the depth limit for the search. Konnector's bidirectional search algorithm is a modification of a standard breadth-first algorithm that tracks the states of two simultaneous breadth-first searches, and alternates between them with the addition of each new edge. Whereas a standard breadth-first search implementation uses a single data structure to track the status of each node (unvisited, previously visited, or exhausted), the bidirectional version uses two such data structures, and requires that the two traversals check each other's node states in order to detect when the traversals overlap.

C. Reconciling Alternate Paths Between Read Pairs

When two paired-end reads are connected by a unique path, the sequence corresponding to the path is used to join the reads into a single pseudoread, as shown in **Fig. 1**. The input sequences that occur before/after the start/goal k-mer of the path search are output unchanged and become the flanking sequences of the pseudoread.

When there are multiple paths between a read pair, a multiple sequence alignment of the paths is performed to determine a consensus sequence, where bases that differ between paths are expressed as IUPAC ambiguity codes [25]. In the current version of Konnector, the multiple sequence alignment is built iteratively using pairwise Smith-Waterman alignment [26], where the first two paths are aligned to build an initial consensus sequence, the third path is aligned to the initial consensus sequence to build a revised consensus sequence, and so on. The order in which the paths are aligned is chosen arbitrarily. In scenarios where alternate paths have different lengths, the multiple sequence alignment is biased toward creating longer sequences. For example, if two paths are identical except for an inserted sequence in one of the paths, the consensus sequence will include the inserted sequence.

The algorithm searches exhaustively for all possible paths between each pair of reads in order to ensure the correctness of the connecting sequences. However, Konnector has several input parameters that may be used to either reduce the computation time or to place tighter constraints on the quality of the results. Most importantly, the user may specify an upper bound on the fragment size (-F option, default 1000) in order to limit the depth of the path search. The user may also limit the breadth of the search by specifying the maximum number of simultaneously active branches (-B option, default 350), in order to curtail costly searches in densely connected areas of the de Bruijn graph. In relation to quality constraints, there are also options for specifying total number of base mismatches allowed across alternate paths (-M option, default 2) and the maximum number of alternate paths that may be merged into a consensus sequence (-P, default 2). On completion, Konnector displays a report detailing the percentages of merges that succeeded and failed due to the various constraints (e.g. too many alternate paths).

III. METHODS

A. Evaluation of Konnector Performance

Human genome chromosome 21 (from the GRCh37 reference) was used to simulate paired-end reads under various base error, depth of coverage and fragment size conditions. Using the sequence simulator pIRS 1.1.1 [27], we created a diploid copy of chromosome 21 (**pirs diploid -i chr21.fa**, with default values) with a heterozygous SNV, indel, and structural variation rates of 0.1%, 0.01% and

TABLE I. **DATA SETS ANALYZED**

Organism	Genome Size	NGS data source	Read length (bp)	Read pairs (M)	Fragment size (bp)	Fold coverage
E.coli K-12	5 Mbp	Simulated	PE100	1.2	400	50X
H.sapiens chromosome 21	48 Mbp	Simulated	PE100	12.0	150-5000	1-60X
C.elegans	97 Mbp	Experimental SRA:ERR294494	PE100	44.7	450	89X
H.sapiens NA19238	3 Gbp	Experimental SRA:ERR309932	PE250	457.0	550	76X

0.0001%, respectively. In duplicate experiments, we simulated paired-end reads from this simulated diploid source (**pirs simulate -i chr21.fa -I chr21copy.fa.snp.indel.inversion.fa -l 100 -x 50 -m 400 -e 0.001**) and tested the effect of the parameter k on Konnector (**abyss-connectpairs -v -v -j 12 -k $k -b 30G -B 300 -F 700**). Conditions such as depth of coverage, base errors and fragment lengths were tested by adjusting the parameters of the read simulator in accordance with the measure tested while setting the k-value to that of the optimal (k=60). Likewise, the effect of Konnector's Bloom filter size (–b option) was adjusted to measure the impact of the false positive rate on the merged yield. Bloom filter sizes ranging from 5 MB to 500 MB were assessed for this purpose, but set high (30 GB) for other runs to maximize the yield.

B. Comparison to Other Pseudo-read Generating Tools

To assess the performance of Konnector, we compared it with two similar tools: GapFiller 2.1.1 and ELOPER 1.2. All experiments were run on our computing cluster running CentOS 5.4, and each read processing job was given a full node with 12 cores and 120 GB of RAM. Simulated data sets for *E. coli* K-12 substrain MG1655 genome (RefSeq accession NC_000913) and human chromosome 21 were included in the comparison to accommodate the ELOPER and GapFiller tools, which have relatively large memory requirements. Attempts to run ELOPER on the human chromosome 21 data set and the two experiment data sets failed due to exhaustion of the available 120 GB of RAM. Similarly, GapFiller was unable to run on the two experimental datasets (*C. elegans* and human genome NA19238) due to insufficient memory.

The pIRS simulator was used on the *E. coli* and human chromosome 21 reference sequences to generate synthetic 100 bp paired-end reads with 0.1% error rate, 50x coverage, and an insert size of 400±50 bp.

Each read extension job was run in triplicates to account for variability between runs. We used the memusg script (https://gist.github.com/netj/526585) to record peak memory usage. For Konnector, we tested a sweep of k-mer values and chose the k-mer size that produced the maximum number of merged reads (best yield). We likewise conducted a sweep for the Bloom filter size parameter of Konnector, in order to determine the minimum Bloom filter size with a false positive rate < 5%. All other parameters were set to default values for the three tools.

ELOPER produced three output files, which we merged into a single file containing only elongated sequences.

All elongated sequences were aligned back to their reference genome using BWA-MEM 0.7.4 [28] in order to calculate the percent genome coverage and pseudo-read accuracy.

C. Assessment of Konnector on Simulated and Experimental Datasets

To evaluate the performance of Konnector on real sequencing data, we downloaded experimental datasets from SRA for *C. elegans* (SRA:ERR294494) and *H. sapiens* NA19238 (SRA:ERR294494), as described in **Table I**. To evaluate the accuracy of the pseudo-reads generated from these reads, we obtained the latest version of the *C. elegans* genome from WormBase (NCBI BioProject PRJNA13758, WS241) and human genome version GRCh37 from the Genome Reference Consortium website for use as reference sequences. The experimental datasets also served to assess the scalability of the tool, with the largest dataset consisting of approximately 1 billion 2 x 250 bp reads (NA19238).

The yield and accuracy of pseudo-reads generated by Konnector were measured across a range of k values for each of the four datasets in **Table I**. For the simulated datasets, only the haploid versions were used; the preparation of these datasets is described in **Section III.A**. For each k, the yield (i.e. percentage of read pairs merged) was counted. The accuracy of the merged sequences was assessed by aligning the pseudo-reads to the reference sequence with BWA-MEM 0.7.4 (with default parameters) and calculating the Levenshtein edit distance of the highest scoring alignment. The Levenshtein edit distance is the minimum number of bases that must be added, deleted, or substituted in the pseudo-read to render it identical to the reference.

IV. RESULTS

A. Konnector Performance

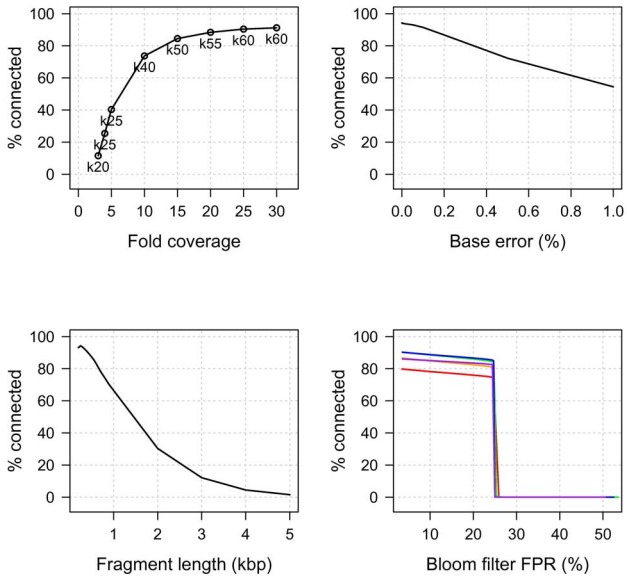

Figure 2. Konnector performance. The performance limit and ideal running conditions of Konnector were assessed using simulated datasets. Unless otherwise specified by the condition under scrutiny, we simulated, in duplicate experiments, 50-fold coverage 0.1% error PE100 reads from 400 bp fragments (~12 M read pairs). The ideal k-value for running Konnector on these data was determined to be 60 bp and a Bloom filter size of 30 GB was used to ensure optimum performance, unless otherwise stated. Each point shows the mean value of the measure. The false positive rate plot (lower right) depicts Konnector runs with k values of 40 (red), 50 (orange), 60 (green), 70 (blue), and 80 (purple) base pairs.

Using simulated datasets, we measured the percentage of read pairs merged by Konnector under varying sequence coverage, sequencing error rates, fragment length, and

Bloom filter false positive rates (**Fig. 2**). For 50x coverage, k=60 bp, base error rate of 0.1%, 400 bp fragment length, and 100 bp paired-end reads, we could connect 91.5% (SD = 0.002%) and 93.0 % (SD = 0.004%) of read pairs from diploid and haploid (not shown) sources, respectively. We could not test the effect of longer reads with pIRS since no error model based on the Illumina sequencing platform was available for cycles >100. However, using the same specifications (50x coverage, 0.1% base error, 30 GB Bloom filter size) and a slightly larger k value (70 bp), with randomly distributed sequence base errors, 98.4% of wgsim-simulated PE150 reads from haploid human chromosome 21 were connected (data not shown).

The yield of connected pairs steadily decreased as we increased the simulated base error rate from 0.0% to 1.0% (**Fig. 2** top right). Given that many error k-mers are filtered out by the two-level Bloom filter, the primary mode of failure was the inability to find error-free start/goal k-mers within the reads to use as the endpoints for the search path. In future work, it may be possible to improve on this aspect of the algorithm by adding error-correcting logic that permutes the bases of k-mers which are not found in the second level Bloom filter.

The yield of Konnector also decreased as we increased the simulated fragment size from 150 to 5000 bp (**Fig. 2,** bottom left). The primary modes of failure in this case were: (i) exceeding our chosen limit of 300 for the maximum breadth of the search (-B option) and (ii) exceeding our chosen limit of 2 for the maximum number of alternate paths (-P option). This result matches our intuition that the cost of

the search and the number of possible paths both increase as the distance between the input reads increases.

With the exception of assessing the Bloom filter FPR (**Fig. 2** bottom right), we allocated the maximum amount of RAM (30 GB) for building the Bloom filter in memory. For a small dataset (12 M PE100 reads, **Table I**), a Bloom filter size of 500 MB was sufficient to connect 90.3% of the pairs with an FPR of 3.5%. It was determined that, with this synthetic dataset, allocating 80 MB of RAM (~24% FPR) was the lowest practical limit to minimize the effect of FPR on yield, and that this critical FPR value seems to be independent of the k-mer size used (**Fig. 2**, lower right panel). The yield for runs with an FPR of approximately 25% or greater dropped abruptly because branching within the graph begins to exceed our chosen maximum breadth limit of 300 for the search (-B option). Twenty five percent is a critical value for FPR because the algorithm tests for four possible neighboring k-mers at each step of the graph traversal; an FPR of 25% implies that the expected number of false neighbors at each k-mer is one, and thus false branches are expected to start from every true k-mer and extend indefinitely. In practice, using all available RAM to build the Bloom filter will ensure a lower false positive rate and better connectivity between read pairs.

B. Read Elongation Tool Comparison

A comparison was performed between the Konnector, ELOPER, and GapFiller read elongation tools using both simulated and experimental reads. Simulated reads were generated from the *E. coli* K-12 and human chromosome 21 reference sequences according to the methods described in **Section III.B**, and real sequencing data was used from the *C. elegans* genome. The results of the comparison for several different performance, completeness, and accuracy metrics are shown in **Table II**.

Although these tools all extend paired-end reads, their approaches differ. Konnector and GapFiller aim to fill the gap between each pair of reads, and thus the length of their output sequences center on the fragment length (400nt). The mean read length and N50 produced by Konnector and GapFiller are similar, as are the results for accuracy and target coverage. However, Konnector generated a higher yield on both simulated data sets (measured as the ratio of number of reads in the output to the number of reads in the input, N_o/N_i).

ELOPER differs from Konnector and GapFiller in the sense that it elongates each read in both directions, and does not necessarily require the read pairs to connect. As a result, the N50 produced by ELOPER is significantly higher on the *E. coli* data set than the other two tools (737 vs. 396 and 406). However, ELOPER did not fare as well with respect to yield and accuracy. The number of reads connected by ELOPER was 60,181, corresponding to 2.59% of the original reads, and the percentage of reads produced by ELOPER that were identical to the reference was only 52.01% in comparison to 94.18% and 95.29% for GapFiller and Konnector, respectively.

The most significant difference between Konnector and the other tools was scalability. While GapFiller ran for 157 hours and required 3.6 GB of memory on the simulated human chromosome 21 dataset, Konnector ran in 1 hour and 22 minutes and required 1.1 GB of RAM. On high-performance computers (12 Intel Xeon CPUs with 120 GB RAM) Konnector was the only tool able to run on the experimental *C. elegans* data set, and did so with a small memory footprint of 1.9 GB.

C. Assessment of Konnector on Simulated and Experimental Datasets

The yield and accuracy of pseudo-reads generated by Konnector were assessed for two simulated datasets and two experimental datasets across a range of k values (**Fig. 3**). The accuracy of the generated pseudo-reads for each dataset was assessed by aligning the pseudo-reads to the reference sequence and calculating the Levenshtein edit distance of the highest scoring alignment.

One limitation of the edit distance calculation used in our assessment was that any IUPAC consensus characters within pseudo-reads were treated as differences regardless of whether the consensus character agreed with the reference or not. As consensus codes are generated by Konnector to represent differences in alternate paths between paired reads, a pseudo-read with an edit distance of one or less was still counted as a valid pseudo-read in the context of diploid experimental datasets, as this position may represent a heterozygous SNP.

The total number of read pairs or pseudo-reads with an edit distance of zero (for haploid data sets) or one (for diploid data sets) after each Konnector run was estimated as:

$$\%Good = \%GoodPseudoreads + (1 - \%Connected) \times \%GoodInputPairs \quad (1)$$

where "%Good" is percentage of output pseudo-reads or unconnected read pairs with edit distance zero or one; "%GoodPseudoreads" is the percentage of pseudo-reads with edit distance zero or one; "%Connected" is the percentage of input read pairs that were merged into pseudo-reads by Konnector; and "%GoodInputPairs" is the percentage of the input read pairs with edit distance zero or one. "%GoodInputPairs" is estimated as the square of the percentage of single end reads with an edit distance of zero, which relies on the simplifying assumption that the edit distances of the first and second reads in a pair are independent and identically distributed.

Equation (1) describes the relationship between the blue, green and black dotted lines in **Fig. 3**. The blue line represents "%Connected", the green line represents "%Good", and the black dotted line represents "%GoodInputPairs". As such, the vertical distance between the green line and the black dotted line of each plot provides an indication of the overall improvement in accuracy achieved by running Konnector on the dataset.

For all four datasets we observe a substantial improvement in the accuracy of the processed reads, with the largest improvements occurring with the simulated datasets for *E. coli* (14% increase at k=60 bp) and human chromosome 21 (15% increase at k=75 bp). We note that there is a general correlation between the Konnector yield and the extent of the accuracy improvement across the datasets; in datasets where a larger proportion of read pairs are connected, a larger gap is seen between the green and black dotted lines. For higher k values, the accuracy of the Konnector output continues to improve even as the yield begins to decline. Thus, using a larger k is important for generating the most accurate pseudo-reads. Moreover, the best choice for k for a given application may be higher than the k value that generates the maximum yield.

We observe that the Konnector yield for the experimental datasets is significantly less than the yield for the simulated datasets. For example, the peak yield for the simulated *E. coli* dataset was 98.3% at k=50 bp versus a peak yield of 78.2% at k=55 bp for the experimental *C. elegans* dataset. This difference cannot be attributed to a larger error rate in the experimental sequencing data, as the position of the black dotted line in the *C. elegans* plot suggests that the error rate is approximately equal to that of the simulated data sets. It is likely that the differing ploidy and repeat content of the genome are important factors in determining the yield across datasets, as they affect the complexity of the de Bruijn graph and the number of alternate paths that exist between read pairs.

The inclusion of the human genome NA19238 data set in **Fig. 3** demonstrates the scalability of Konnector, as this data set consists of 914 million reads and provides approximately 76-fold coverage of the 3 GB human genome. To run Konnector on NA19238, we first built a 20 GB bloom filter

file on a single machine with 48 GB of RAM, using the abyss-bloom utility provided with Konnector. We then partitioned the read pairs evenly across 20 machines, each with 48 GB RAM, and ran parallel instances of Konnector on each host. The overall running time for the job was approximately four and a half days.

The source code for the exact version of Konnector evaluated here is available from https://github.com/bcgsc/abyss/tree/konnector-prelease; please note that in this version, the executable is named "abyss-connectpairs" rather than "konnector".

TABLE II. READ ELONGATION TOOL COMPARISON

	***E. coli* (synthetic)**			***H. sapiens chr 21* (synthetic)**			***C. elegans***		
	ELOPER	***GapFiller***	***Konnector***	***ELOPER***	***GapFiller***	***Konnector***	***ELOPER***	***GapFiller***	***Konnector***
time (hms)	10m46s	32m39s	6m15s	exceeds available memory (120GB)	157h7m42s	1h22m55s	exceeds available memory (120GB)	exceeds available memory (120GB)	5h5m21s
peak mem (MB)	19,013	476	81		3,735	1,151			1,954
altered read (>= 100nt)	60,181	674,115	1,128,604		9,817,452	11,079,871			68,879,932
mean read length (>= 100nt)	414	393	399		398	399			467
N50	737	396	406		400	400			475
N_{out}/N_{in} (bases)	0.14	1.14	1.94		1.62	1.84			1.80
N_{out}/N_{in} (reads)	0.026	0.290	0.486		0.41	0.46			0.385
% genome coverage	99.73	99.63	99.72		72.01	72.63			98.71
% reads identical to reference	52.01	94.18	95.29		93.00	96.96			94.08*

* Pseudoreads with an edit distance of one or less were counted as identical to the reference, in order to accommodate heterozygous SNPs occurring within the diploid *C. elegans* genome.

V. FUTURE WORK

In this paper, we have demonstrated that Konnector produces results with equal or better yield and quality as two existing tools, ELOPER[14] and GapFiller[15], while requiring substantially less memory and computation time. Below, we discuss some possible applications for the technology.

A. De Novo Genome Assembly

The ability of read-merging tools to improve the quality and contiguity of *de novo* assemblies has been previously demonstrated for both overlapping [12, 13] and non-overlapping reads [14, 15]. From our preliminary work using Konnector pseudoreads as input to the ABySS assembler [1], we observed marked improvement at the initial assembly stage (ie. de Bruijn graph), namely a higher contiguity and lower sequence count, which is expected since larger assembly k values are available for k-mer assembly (not shown). In future work, we will evaluate the utility of Konnector for improving *de novo* assemblies.

B. Gap Filling

The scaffolding stage [29] of *de novo* assembly results in sequences that are joined by gaps of varying size, conventionally represented by runs of "N" characters. Previously published gap-filling tools have successfully employed a local assembly based approach to fill scaffold gaps, where the reads for the local assembly are recruited by paired-end alignments to the gap region [30, 31]. Konnector should be readily adaptable to gap filling by substituting the flanking sequences of the gaps in place of the input reads, and we plan to evaluate this application in future work.

C. Variant Calling

While the majority of existing tools for SNP calling and structural variant calling rely on aligning reads to the reference (e.g. [32]), tools that employ a *de novo* assembly based approach such as [33, 34] have advantages in situations where a reference is not available or where the reads may be highly divergent from the reference.

As Konnector searches for all connecting paths between each pair of input reads, it may be possible to use the differences between alternate paths to identify genomic variants. For example, Konnector uses IUPAC codes to represent single base differences across paths, and this information can potentially be used downstream to identify single nucleotide variants (SNVs). As an initial test of this idea, we profiled 34,884 SNVs simulated in *H. sapiens* chromosome 21, using 100 bp reads on 400 bp fragments, 60-fold coverage, and 0.1% base error. We were able to recover 33,516 of the 34,884 (96.1%) SNVs and found that no erroneous consensus codes were reported in the pseudoreads. While this approach is simplistic in comparison to existing tools that incorporate allelic frequencies and statistical models to perform SNP calling (e.g. [32]), it may be possible to adapt Konnector into a more sophisticated tool with further work. For example, a counting Bloom filter [35] could be used to track allelic frequencies in lieu of the two-stage Bloom filter described here.

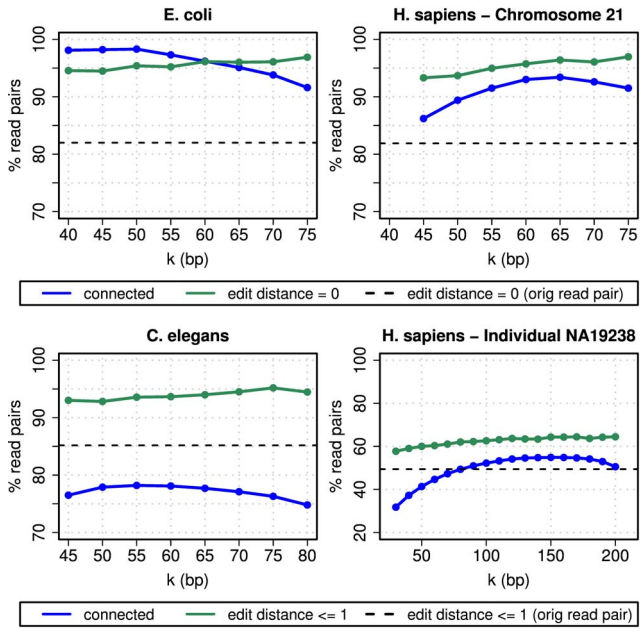

Figure 3. Yield and accuracy assessment of Konnector on four data sets: *E. coli* (sythetic, top left), *H. sapiens* chromosome 21 (synthetic, top right), *C. elegans* (experimental, bottom left), and *H. sapiens* individual NA19238 (experimental, bottom right). Each plot shows the percentage of read pairs connected (blue), the percentage pseudo-reads and unconnected reads with an edit distance of zero or one from the reference (green), and the percentage of input read pairs with a edit distance of zero or one from the reference (black, dotted). The vertical distance between the green and black dotted lines in each plot serves as an indication of the accuracy gains achieved by running Konnector on the dataset. The simulated *E. coli* data set produced the best yield of 98.3%, whereas the largest improvement in accuracy occurred in the simulated human chromosome 21 dataset with an increase of 15% at k=75.

VI. CONCLUSION

Merging overlapping paired-end reads has become common practice before alignment and assembly. For the case where paired-end reads do not overlap, we have designed Konnector, a tool for filling the gap between any given paired-end reads. It accomplishes the task quickly, efficiently in low memory, and correctly. To our knowledge, it is the only application of its kind capable of representing an entire de Bruijn k-mer graph of over 900 million paired-end reads on compute hardware with 48 GB of RAM.

ACKNOWLEDGMENT

The authors thank the funding organizations, Genome Canada, British Columbia Cancer Foundation, and Genome British Columbia for their partial support. Research reported in this publication was also partly supported by the National Human Genome Research Institute of the National Institutes of Health under Award Number R01HG007182. The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Institutes of Health or other funding organizations.

REFERENCES

[1] J. T. Simpson, K. Wong, S. D. Jackman, J. E. Schein, S. J. Jones, and I. Birol, "ABySS: a parallel assembler for short read sequence data," *Genome Res,* vol. 19, pp. 1117-23, Jun 2009.

[2] D. R. Zerbino and E. Birney, "Velvet: Algorithms for de novo short read assembly using de Bruijn graphs," *Genome Research,* vol. 18, pp. 821-829, May 2008.

[3] J. Butler, I. MacCallum, M. Kleber, I. A. Shlyakhter, M. K. Belmonte, E. S. Lander, C. Nusbaum, and D. B. Jaffe, "ALLPATHS: de novo assembly of whole-genome shotgun microreads," *Genome Research,* vol. 18, pp. 810-20, May 2008.

[4] J. T. Simpson and R. Durbin, "Efficient de novo assembly of large genomes using compressed data structures," *Genome Research,* vol. 22, pp. 549-556, Mar 2012.

[5] N. Donmez and M. Brudno, "SCARPA: scaffolding reads with practical algorithms," *Bioinformatics,* vol. 29, pp. 428-434, Feb 2013.

[6] M. Boetzer, C. V. Henkel, H. J. Jansen, D. Butler, and W. Pirovano, "Scaffolding pre-assembled contigs using SSPACE," *Bioinformatics,* vol. 27, pp. 578-579, Feb 15 2011.

[7] M. Pop, D. S. Kosack, and S. L. Salzberg, "Hierarchical scaffolding with Bambus," *Genome Research,* vol. 14, pp. 149-159, Jan 2004.

[8] K. Chen, J. W. Wallis, M. D. McLellan, D. E. Larson, J. M. Kalicki, C. S. Pohl, S. D. McGrath, M. C. Wendl, Q. Zhang, D. P. Locke, X. Shi, R. S. Fulton, T. J. Ley, R. K. Wilson, L. Ding, and E. R. Mardis, "BreakDancer: an algorithm for high-resolution mapping of genomic structural variation," *Nat Methods,* vol. 6, pp. 677-81, Sep 2009.

[9] K. Ye, M. H. Schulz, Q. Long, R. Apweiler, and Z. Ning, "Pindel: a pattern growth approach to detect break points of large deletions and medium sized insertions from paired-end short reads," *Bioinformatics,* vol. 25, pp. 2865-2871, Nov 1 2009.

[10] S. Lee, F. Hormozdiari, C. Alkan, and M. Brudno, "MoDIL: detecting small indels from clone-end sequencing with mixtures of distributions," *Nature Methods,* vol. 6, pp. 473-474, Jul 2009.

[11] T. Rausch, T. Zichner, A. Schlattl, A. M. Stuetz, V. Benes, and J. O. Korbel, "DELLY: structural variant discovery by integrated paired-end and split-read analysis," *Bioinformatics,* vol. 28, pp. I333-I339, Sep 15 2012.

[12] T. Magoc and S. L. Salzberg, "FLASH: fast length adjustment of short reads to improve genome assemblies," *Bioinformatics,* vol. 27, pp. 2957-63, Nov 1 2011.

[13] B. Liu, J. Yuan, S.-M. Yiu, Z. Li, Y. Xie, Y. Chen, Y. Shi, H. Zhang, Y. Li, T.-W. Lam, and R. Luo, "COPE: an accurate k-mer-based pair-end reads connection tool to facilitate genome assembly," *Bioinformatics,* vol. 28, pp. 2870-2874, Nov 15 2012.

[14] D. H. Silver, S. Ben-Elazar, A. Bogoslavsky, and I. Yanai, "ELOPER: elongation of paired-end reads as a pre-processing tool for improved de novo genome assembly," *Bioinformatics,* vol. 29, pp. 1455-1457, Jun 1 2013.

[15] F. Nadalin, F. Vezzi, and A. Policriti, "GapFiller: a de novo assembly approach to fill the gap within paired reads," *Bmc Bioinformatics,* vol. 13, Sep 7 2012.

[16] A. V. Zimin, G. Marcais, D. Puiu, M. Roberts, S. L. Salzberg, and J. A. Yorke, "The MaSuRCA genome assembler," *Bioinformatics,* vol. 29, pp. 2669-2677, Nov 1 2013.

[17] S. Boisvert, F. Laviolette, and J. Corbeil, "Ray: simultaneous assembly of reads from a mix of high-throughput sequencing technologies," *J Comput Biol,* vol. 17, pp. 1519-33, Nov 2010.

[18] H. Li and R. Durbin, "Fast and accurate short read alignment with Burrows-Wheeler transform," *Bioinformatics,* vol. 25, pp. 1754-60, Jul 15 2009.

[19] B. Langmead, C. Trapnell, M. Pop, and S. L. Salzberg, "Ultrafast and memory-efficient alignment of short DNA sequences to the human genome," *Genome Biol,* vol. 10, p. R25, 2009.

[20] H. Stranneheim, M. Kaller, T. Allander, B. Andersson, L. Arvestad, and J. Lundeberg, "Classification of DNA sequences using Bloom filters," *Bioinformatics,* vol. 26, pp. 1595-600, Jul 1 2010.

[21] B. H. Bloom, "Space/Time Tradeoffs in Hash Coding With Allowable Errors," *Communications of the Acm,* vol. 13, pp. 422-&, 1970 1970.

[22] J. Pell, A. Hintze, R. Canino-Koning, A. Howe, J. M. Tiedje, and C. T. Brown, "Scaling metagenome sequence assembly with probabilistic de Bruijn graphs," *Proceedings of the National Academy of Sciences of the United States of America,* vol. 109, pp. 13272-13277, Aug 14 2012.

[23] R. Chikhi and G. Rizk, "Space-efficient and exact de Bruijn graph representation based on a Bloom filter," *Algorithms for Molecular Biology,* vol. 8, Sep 16 2013.

[24] I. Birol, A. Raymond, S. D. Jackman, S. Pleasance, R. Coope, G. A. Taylor, M. M. Yuen, C. I. Keeling, D. Brand, B. P. Vandervalk, H. Kirk, P. Pandoh, R. A. Moore, Y. Zhao, A. J. Mungall, B. Jaquish, A. Yanchuk, C. Ritland, B. Boyle, J. Bousquet, K. Ritland, J. Mackay, J. Bohlmann, and S. J. Jones, "Assembling the 20 Gb white spruce (Picea glauca) genome from whole-genome shotgun sequencing data," *Bioinformatics,* May 22 2013.

[25] A. Cornishbowden, "Nomenclature For Incompletely Specified Bases In Nucleic-Acid Sequences - Recommendations 1984," *Nucleic Acids Research,* vol. 13, pp. 3021-3030, 1985 1985.

[26] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *J Mol Biol,* vol. 147, pp. 195-7, Mar 25 1981.

[27] X. Hu, J. Yuan, Y. Shi, J. Lu, B. Liu, Z. Li, Y. Chen, D. Mu, H. Zhang, N. Li, Z. Yue, F. Bai, H. Li, and W. Fan, "pIRS: Profile-based Illumina pair-end reads simulator," *Bioinformatics,* vol. 28, pp. 1533-5, Jun 1 2012.

[28] H. Li, "Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM," *arXiv preprint,* 2013.

[29] M. N. Hunt, Chris; Berriman, Matthew; Otto, Thomas D., "A comprehensive evaluation of assembly scaffolding tools," *Genome Biol,* vol. 15, p. R42, 2014.

[30] M. Boetzer and W. Pirovano, "Toward almost closed genomes with GapFiller," *Genome Biol,* vol. 13, p. R56, 2012.

[31] R. Luo, B. Liu, Y. Xie, Z. Li, W. Huang, J. Yuan, G. He, Y. Chen, Q. Pan, Y. Liu, J. Tang, G. Wu, H. Zhang, Y. Shi, Y. Liu, C. Yu, B. Wang, Y. Lu, C. Han, D. W. Cheung, S.-M. Yiu, S. Peng, Z. Xioqian, G. Liu, X. Liao, Y. Li, H. Yang, J. Wang, T.-W. Lam, and J. Wang, "SOAPdenovo2: an empirically improved memory-efficient short-read de novo assembler," *GigaScience,* vol. 1, 2012.

[32] M. A. DePristo, E. Banks, R. E. Poplin, K. V. Garimella, J. R. Macguire, C. Hartl, A. A. Philippakis, G. d. Angel, M. A. Rivas, M. Hanna, A. McKenna, T. J. Fennell, A. M. Kernytsky, A. Y. Sivachenko, K. Cibulskis, S. B. Gabriel, D. Altshuler, and M. J. Daly, "A framework for variation discovery and genotyping using next-generation DNA sequencing data," *Nat Genet,* vol. 43, p. 8, 2011.

[33] Z. C. Iqbal, Mario; Turner, Isaac; Flicek, Paul; McVean, Gil, "De novo assembly and genotyping of variants using colored de Bruijn graphs," *Nature genetics,* vol. 44, p. 7, 2012.

[34] G. Robertson, J. Schein, R. Chiu, R. Corbett, M. Field, S. D. Jackman, K. Mungall, S. Lee, M. O. Hisanga, J. Q. Qian, M. Griffith, A. Raymond, N. Thiessen, T. Cezard, Y. S. Butterfield, R. Newsome, S. K. Chan, R. She, R. Varhol, B. Kamoh, A.-L. Prabhu, A. Tam, Y. Zhao, R. A. Moore, M. Hirst, M. A. Marra, S. J. M. Jones, P. A. Hoodless, and I. Birol, "De novo assembly and analysis of RNA-seq data," *Nature Methods,* vol. 7, p. 4, 2010.

[35] L. C. Fan, Pei; Almeida, Jussara; Broder, Andrei Z. , "Summary cache: a scalable wide-area web cache sharing protocol," *IEEE/ACM Transactions on Networking (TON),* vol. 8, p. 13, 2000.