Bioinformatics, 2015, 1–9
doi: 10.1093/bioinformatics/btv311
Advance Access Publication Date: 20 May 2015
Original Paper

OXFORD

Sequence analysis

When less is more: 'slicing' sequencing data improves read decoding accuracy and *de novo* assembly quality

Stefano Lonardi[1,*], Hamid Mirebrahim[1], Steve Wanamaker[2], Matthew Alpert[1], Gianfranco Ciardo[3], Denisa Duma[1,4] and Timothy J. Close[2,*]

[1]Department of Computer Science and Engineering, [2]Department of Botany and Plant Sciences, University of California, Riverside, CA 92521, [3]Department of Computer Science, Iowa State University, Ames, IA 50011 and [4]Baylor College of Medicine, Houston, TX 77030, USA

*To whom correspondence should be addressed.

Associate Editor: John Hancock

Received on January 5, 2015; revised on May 6, 2015; accepted on May 13, 2015

Abstract

Motivation: As the invention of DNA sequencing in the 70s, computational biologists have had to deal with the problem of *de novo* genome assembly with limited (or insufficient) depth of sequencing. In this work, we investigate the opposite problem, that is, the challenge of dealing with excessive depth of sequencing.

Results: We explore the effect of ultra-deep sequencing data in two domains: (i) the problem of decoding reads to bacterial artificial chromosome (BAC) clones (in the context of the combinatorial pooling design we have recently proposed), and (ii) the problem of *de novo* assembly of BAC clones. Using real ultra-deep sequencing data, we show that when the depth of sequencing increases over a certain threshold, sequencing errors make these two problems harder and harder (instead of easier, as one would expect with error-free data), and as a consequence the quality of the solution degrades with more and more data. For the first problem, we propose an effective solution based on 'divide and conquer': we 'slice' a large dataset into smaller samples of optimal size, decode each slice independently, and then merge the results. Experimental results on over 15 000 barley BACs and over 4000 cowpea BACs demonstrate a significant improvement in the quality of the decoding and the final assembly. For the second problem, we show for the first time that modern *de novo* assemblers cannot take advantage of ultra-deep sequencing data.

Availability and implementation: Python scripts to process slices and resolve decoding conflicts are available from http://goo.gl/YXgdHT; software Hashfilter can be downloaded from http://goo.gl/MlyZHs

Contact: stelo@cs.ucr.edu or timothy.close@ucr.edu

Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

We have recently introduced in (Lonardi *et al.*, 2013) a novel protocol for clone-by-clone *de novo* genome sequencing that leverages recent advances in combinatorial pooling design (also known as *group testing*). In our sequencing protocol, subsets of non-redundant genome-tiling bacterial artificial chromosomes (BACs) are chosen to form intersecting pools, then groups of pools are sequenced on an Illumina sequencing instrument via low-multiplex (DNA

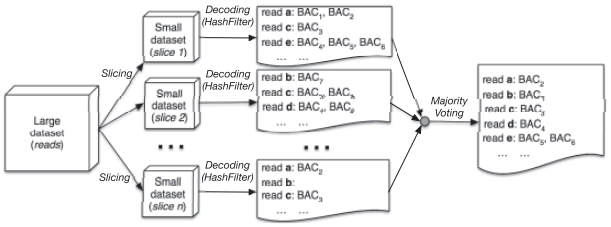

Fig. 1. An illustration of the strategy to improve read decoding: (i) a large dataset of reads to be decoded is "sliced" in n smaller datasets of optimal size, (ii) each slice is decoded independently and (iii) read-to-BAC assignments for each slice are merged and conflicts are resolved

barcoding). Sequenced reads can be assigned/decoded to specific BACs by relying on the combinatorial structure of the pooling design: since the identity of each BAC is encoded within the pooling pattern, the identity of each read is similarly encoded within the pattern of pools in which it occurs. Finally, BACs are assembled individually, simplifying the problem of resolving genome-wide repetitive sequences.

In (Lonardi *et al.*, 2013), we reported preliminary assembly statistics on the performance of our protocol in four barley (*Hordeum vulgare*) BAC sets (Hv3–Hv6). Further analysis on additional barley BAC sets and two genome-wide BAC sets for cowpea (*Vigna unguiculata*) revealed that the raw sequence data for some datasets was of significantly lower quality (i.e. higher sequencing error rate) than others. We realized that our decoding strategy, solely based on the software HashFilter (Lonardi *et al.*, 2013), was insufficient to deal with the amount of noise in poor quality datasets. We attempted to (i) trim/clean the reads more aggressively or with different methods, (ii) identify low quality tiles on the flow cell and remove the corresponding reads (e.g. tiles on the 'bottom middle swath'), (iii) identify positions in the reads possibly affected by sequencing 'bubbles' and (iv) post-process the reads using available error-correction software tools (e.g. Quake, Reptile). Unfortunately, none of these steps accomplished a dramatic increase in the percentage of reads that could be assigned to BACs, indicating that the quality of the dataset did not improve very much. These attempts to improve the outcome led however, to a serendipitous discovery: we noticed that when HashFilter processed only a portion of the dataset, the proportion of assigned/decoded reads increased. This observation initially seemed counterintuitive: we expected that feeding less data into our algorithm meant that we had less information to work with, thus decrease the decoding performance. Instead, the explanation is that when data is corrupted, more (noisy) data is not better, but worse.

The study reported here directly addresses the observation that when dealing with large quantities of imperfect sequencing data, 'less' can be 'more'. More specifically, we report (i) an extensive analysis of the trade off between the size of the datasets and the ability of decoding reads to individual BACs; (ii) a method based on 'slicing' datasets that significantly improves the number of decoded reads and the quality of the resulting BAC assemblies; (iii) an analysis of BAC assembly quality as a function of the depth of sequencing, for both real and synthetic data. Our algorithmic solution relies on a divide-and-conquer approach, as illustrated in Figure 1.

2 Methods

2.1 Pooling design

We applied the combinatorial pooling scheme described in (Lonardi *et al.*, 2013) to BAC clones for (i) a gene-enriched portion of the genome of *H.vulgare L.* (barley), and (ii) the whole genome of *V.unguiculata* (cowpea). Briefly, in our sequencing protocol we (i) obtain a BAC library for the target organism; (ii) select gene-enriched BACs from the library (optional); (iii) fingerprint BACs and build a physical map; (iv) select a minimum tiling path (MTP) from the physical map; (v) pool the MTP BACs according to the shifted transversal design; (vi) sequence the DNA in each pool, trim/clean sequenced reads; (vii) assign reads to BACs (*deconvolution*); (viii) assemble reads BAC-by-BAC using a short-read assembler.

We should first note that a rough draft of the ≈5300 Mb barley genome is now available (Stein *et al.*, 2012): our BAC sequencing work had contributed to that effort, but is distinct. In our work, we focused on the gene-enriched portion of the genome (Muñoz-Amatriaín *et al.*, 2015). We started with a 6.3× genome equivalent barley BAC library which contains 313 344 BACs with an average insert size of 106 kb (Yu *et al.*, 2000). About 84 000 gene-enriched BACs were identified and fingerprinted using high-information-content fingerprinting (Luo *et al.*, 2003; Muñoz-Amatriaín *et al.*, 2015). From the fingerprinting data a physical map was produced (Bozdag *et al.*, 2007; Soderlund *et al.*, 2000) and a MTP of about 15 000 clones was derived (Bozdag *et al.*, 2013; Muñoz-Amatriaín *et al.*, 2015). Seven sets of $n = 2197$ clones were chosen to be pooled according to the *shifted transversal design* (Thierry-Mieg, 2006), which we called Hv3, Hv4, …, Hv9 (Hv1 and Hv2 were pilot experiments). An additional set of $n = 1053$ clones (called Hv10) was pooled using the shifted transversal design with different pooling parameters (see below).

A pooling scheme based on the *shifted transversal design* (Thierry-Mieg, 2006), is defined by (P, L, Γ), where P is a prime number, L defines the number of layers and Γ is a small integer. A *layer* is one of the classes in the partition of BACs and consists of exactly P pools: the larger the number of layers, the higher is the *decodability*. The decodability of the pooling design determines what is the largest number of 'positive' objects that can be decoded: in our case, a d-decodable pooling design will handle the overlap of at most d MTP clones. By construction the total number of pools is $P \times L$. If we set Γ to be the smallest integer such that $P^{\Gamma+1} \geq N$ where N is the number of BACs that need to be pooled, then the decodability of the design is $\lfloor (L-1)/\Gamma \rfloor$.

For barley sets Hv3, Hv4, …, Hv9, we chose parameters $P = 13$, $L = 7$ and $\Gamma = 2$, so that we could handle $P^{\Gamma+1} = 2197$ samples and make the scheme $\lfloor (L-1)/\Gamma \rfloor = 3$-decodable. We expected each non-repetitive read to belong to at most two BACs if the MTP had been computed perfectly, or rarely three BACs when considering imperfections, so we set $d = 3$. Each of the $L = 7$ layers consisted of $P = 13$ pools, for a total of 91 BAC pools. In this pooling design, each BAC is contained in $L = 7$ pools and each pool contains $P^{\Gamma} = 169$ BACs. We call the set of L pools to which a BAC is assigned, the *BAC signature*. Any two BAC signatures can share at most $\Gamma = 2$ pools, and any three BAC signatures can share at most $3\Gamma = 6$ pools. For sets Hv3–Hv8, Vu1 and Vu2, we manually pooled 2197 BACs thus exhausting all the 'available' signature for the pooling design. However, for set Hv9 we only used 1717 signatures. Set Hv10 was pooled using a different design: we chose pooling parameters $P = 11$, $L = 7$ and $\Gamma = 2$, for a total of $P^{\Gamma+1} = 1331$ BAC signatures, however, we only used 1053 signatures. BAC signatures that were available but not used in the pooling were called *ghosts*.

Cowpea's genome size is estimated at 620 Mb and it is yet to be fully sequenced. For cowpea we started from a 17× depth of coverage BAC library containing about 60 000 BACs from the African breeding genotype IT97K-499-35 with an average insert size of 150 kb. Cowpea BACs were fingerprinted using high information

content fingerprinting (Ding *et al.*, 2001; Luo *et al.*, 2003). A physical map was produced from 43 717 fingerprinted BACs with a depth of 11 × genome coverage (Bozdag *et al.*, 2007; Soderlund *et al.*, 2000), and a MTP comprised of 4394 clones was derived (Bozdag *et al.*, 2013). The set of MTP clones was split in two sets of $n = 2197$ BACs (called hereafter Vu1 and Vu2), each of which was pooled according to the shifted transversal design (Thierry-Mieg, 2006), with the same pooling parameters used for Hv3–Hv9.

To take advantage of the high throughput of sequencing of the Illumina HiSeq2000, 13–20 pools in each set were multiplexed on each lane, using custom multiplexing adapters. After the sequenced reads in each lane were demultiplexed, we obtained an average of 1764 million reads in each set with a read length of about 92 bases and an insert size of 275 bases. Reads were quality-trimmed and cleaned of spurious sequencing adaptors, and then reads affected by *Escherichia coli* contamination or BAC vector were discarded. The percentage of *E.coli* contamination averaged around 43%: as a consequence, the average number of usable reads after quality trimming and cleaning decreased to about 824 million, with an average high-quality read length of about 89 bases. Supplementary Table S1 reports the number of reads, number of bases, average read length and *E.coli* contamination for each of the 10 sets (Hv3, Hv4, Hv5, Hv6, Hv7, Hv8, Hv9, Hv10, Vu1 and Vu2). Raw reads for barley and cowpea BACs have been deposited in NCBI SRA accession number SRA051780, SRA051535, SRA051768, SRA073696, SRA051739 (barley); SRA052227 and SRA052228 (cowpea).

2.2 Read decoding analysis

The 91 pools (77 for Hv10) of trimmed reads for barley and cowpea were processed using our *k*-mer based algorithm called HashFilter, which is fully described in (Lonardi *et al.*, 2013). Briefly, HashFilter builds a hash table of all distinct *k*-mers in the 91 (or 77) pools of reads, and records for each *k*-mer the set of pools where it occurs. Then it processes each read individually: (i) a read r is decomposed in its constitutive *k*-mers; (ii) the set of pools of each *k*-mer is fetched from the hash table, and matched against the BAC signatures (allowing for a small number of missing/extra pools); (iii) the union of *k*-mer signatures that match a valid BAC signature determines the BAC assignment for read r. Recall that since our pooling is 3-decodable, each read can be assigned to 0–3 BACs.

For some of the datasets, the percentage of reads decoded using this procedure was very low. For instance HashFilter could decode only 23.8% of the reads in Hv9. We suspected a higher percentage of sequencing errors in Hv9 compared with previous datasets, so we conducted many experiments to improve the decoding performance on this dataset, including (i) tweaking the parameters and the algorithm in HashFilter, (ii) correcting the reads using Quake and Reptile, (iii) increasing the stringency for quality values in the trimming step, (iv) considering only reads that appeared exactly at least twice, (v) using on the left or the right read (for paired-end reads). None of these actions increased the number of decoded reads in Hv9 > 36.6%, which was still unsatisfactory. To our initial surprise, running HashFilter on a fraction of the reads yielded higher decoding percentages, which suggested the idea to 'slice' the data.

We remind the reader that HashFilter has the ability to ignore *k*-mers affected by sequencing errors: if the number t of non-zero counts of a *k*-mer signature belongs to the interval $[L+1, 2L-\Gamma-1]$, HashFilter removes from the *k*-mer signature the $t-L$ pools with the lowest counts [for details, see Case 4 and 6 of step G in Lonardi *et al.*, (2013)]. If one assumes that *k*-mers with sequencing errors are rarer than error-free *k*-mers, spurious pools will have a low *k*-mer count and will be removed before the reads are decoded. In addition to this feature, HashFilter also has the option to disregard entirely a *k*-mer that appears rarely, which is likely to contain sequencing errors.

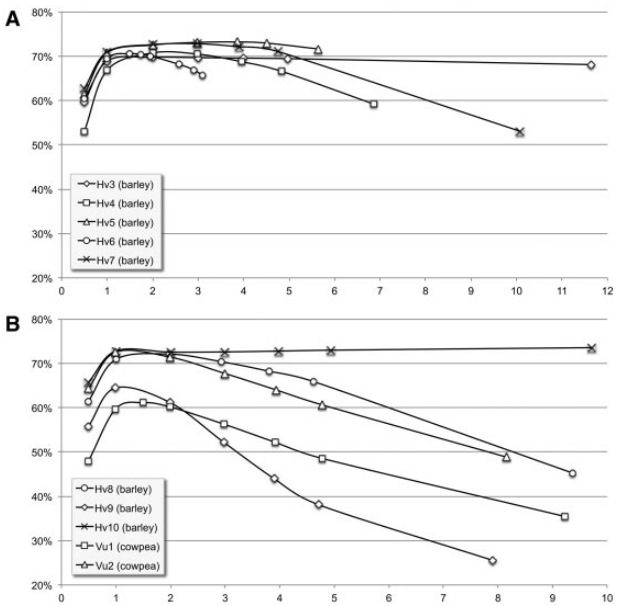

Fig. 2. The percentage of reads decoded by HashFilter ($k = 26$) on dataset **(A)** Hv3, Hv4, Hv5, Hv6 and Hv7 **(B)** Hv8, Hv9, Hv10, Vu1, and Vu2 as a function of the number of reads given in input (*x*: number of million of reads sampled in each dataset)

The next question was to study the dependency between the size of the dataset and the performance of the decoding algorithm. To this end, we took samples of the original 91 (or 77) set of reads in sizes of 0.5, 1, 2, 3, 4 and 5 M reads (details on the sampling method can be found in the next section) and computed the percentage of reads decoded by HashFilter on these samples of increasing sizes. Figure 2A shows the percentages of decoded reads for sets Hv3, Hv4, Hv5, Hv6 and Hv7; Figure 2B is for Hv8, Hv9, Hv10, Vu1 and Vu2. The *x*-axis is the number of reads per pool (in millions) given in input to HashFilter ($k = 26$). The rightmost point on these graphs corresponds to the full dataset. The datasets used to generate Figure 2 are available as Supplementary Dataset 1.

Several observations on Figure 2 are in order. First, observe that when the number of reads per pool is too small (0.5–1 M) the percentage of reads decoded by HashFilter is low. Similarly, when the number of reads per pool is large, the percentage of reads decoded by HashFilter can be low for some datasets. We believe that when the input size is small, there is not enough information in the hash table of *k*-mers to accurately decode the reads. However, when the input size is large, sequencing errors in the data introduce spurious *k*-mers in the hash table, which has the effect of deteriorating HashFilter's decoding performance. Observe that almost all these curves reach a maximum in the range 1–3 M reads. For datasets whose 'optimal number' of reads is low, we can speculate the amount of sequencing error to be higher. Also observe the large variability among these 10 datasets. At one extreme, graphs for Hv3, Hv10 and Hv5 are very 'flat' indicating low sequencing errors; at the other extreme, graphs for Vu1 and Vu2 degrade very quickly after the peak, indicating poorer data quality.

We also carried out a simulation study using synthetic reads generated from the rice genome (*Oryza sativa*). For this simulation we started from an MTP containing 3827 BACs with an average length

of about 150 kb, which spanned 91% of the rice genome (which is about 390 Mb). We pooled *in silico* a subset of 2197 BACs from the set above according to the shifted transversal design [see Lonardi *et al.*, (2013) for details]. We generated 2 M synthetic reads using WGSim (github.com/lh3/wgsim) for each of the 91 resulting rice BAC pools. Reads were 104 bases long with 1% sequencing error rate (no insertions and deletions errors were allowed). A total of 208 Mbp gave an expected 56× coverage for each BAC. We ran HashFilter on the read datasets in slices of 0.25, 0.5, 1, 1.5 and 2 M (full dataset). The percentage of decoded reads (see Supplementary Fig. S1) peaks at 1.5 M, and mirrors the observations made on real data. Even for synthetic reads, more data does not necessarily imply improved decoding performance.

2.3 Improved decoding algorithm

Our improved decoding algorithm first executes HashFilter on progressively larger samples of the dataset (e.g. 0.5, 1, 2, 3, 4, 5 M and full dataset) for a given value of k. Our sampling algorithm selects reads uniformly at random along the input file: taking a prefix of the dataset is not a good idea because reads in the file are organized according to their spatial organization on the flowcell, possibly introducing biases.

When the sample size is greater than the pool size, the entire pool is used for decoding. Otherwise, reads in pools larger than the sample size are uniformly sampled in order to meet the sample size constraint. As a result of this process, the size of each pool in a 'slice' will be *at most* the sample size, but some of the pools will be smaller. The objective is to find the sample size that maximizes the number of reads decoded by HashFilter.

We observed that the optimal value of the sample size is somewhat independent from k as long as it is chosen 'reasonably large', say $k > 20$ for large eukaryotic genomes. Supplementary Figure S2 illustrates that running HashFilter with $k = 20, 23, \ldots, 32$ gives rise to parallel curves. One can save time by running HashFilter with smaller values of k in order to find the optimal data size.

Once the optimal sample size n is determined, the algorithm finds the size m of the largest pool to calibrate d datasets (hereafter called *slices*) each one of which has at most n reads per pool. For instance, if the optimal slice size is $n = 2$ M reads, and the largest pool has $m = 10$ M reads, the algorithm will create $d = m/n = 5$ slices: each one will be composed of 91 pools, each of which has at most 2 M reads. Observe that the number of reads in each pool can vary significantly. For instance in Hv3, the largest pool has almost 23 M reads, and the smallest has about 3 M reads. Smaller pools will contribute their reads to multiple slices. For instance, if there is a pool of size 2 M in the same example described earlier, these reads will appear in all five slices. In general, if a pool size is $\leq n$, the entire pool will be used in each slice.

Then, the algorithms run HashFilter d times, once on each of the d slices—which involves creating d individual hash tables. For this step, we recommend using the largest possible value of k ($k = 32$), because the percentage of decoded reads for a given input size increases with k (see Supplementary Fig. S2). Then, the algorithm merges the d independent HashFilter's outputs. If a read is decoded in only one slice, it will be simply copied in the output. If a read is decoded multiple times in different slices and the independent decodings do not agree, a conflict resolution step is necessary. In our running example, reads in the small 2 M-reads pool will be decoded five times: it is possible that HashFilter will assign a read to five different BAC sets. In order to identify reads decoded multiple times, our algorithm first concatenates the d text outputs of HashFilter, then sorts the reads by their unique identifier (ID), so that reads with the same ID are consecutive in the file. Recall that HashFilter assigns each read to a set composed of 0–3 BACs. A *group* is the set of all BAC (assignment) sets for a single-end read. When a read is paired-end, we have a *left group* for the left read and a *right group* for the right read. For instance in Figure 1, single-end read **c** is decoded by HashFilter at least three times: in slice 1 read **c** is assigned to BAC_3, in slice 2 it is assigned to BAC_3 and BAC_8, and in slice n it is assigned to BAC_3. The set $\{\{BAC_3\}, \{BAC_3, BAC_8\}, \{BAC_3\}\}$ is the group for read **c**. If a read has been decoded at least twice by HashFilter and the sets in its group are not identical, the following algorithm computes the most likely assignment according to a set of rules which are checked in order (i.e. the first one that applies is used, and subsequent rules are not considered).

i. if a read is single-end and its group contains one or more BACs which have 75%-majority or higher, then the read is assigned to those majority BAC(s);
ii. if a read is paired-end, and both its left group and its right group are non-empty, and the union of the left and the right group contains one or more BACs which have 50%-majority or higher, then both the left and the right read are assigned to those majority BAC(s);
iii. if a read is paired-end, and either its left or its right group are empty, and the non-empty group contains one or more BACs which have 75%-majority or higher, then both the left and the right read are assigned to those majority BAC(s);
iv. if a read is paired-end, and its left group is not identical to its right group, then both the left and the right read are not assigned.

In the example on read **c**, since BAC_3 is has 100%-majority (appears in all three assignments) but BAC_8 has only 33%-majority (appears in one of the three assignments), we assign read **c** to BAC_3 but not to BAC_8.

3 Results

Once all the decoded reads are assigned to 1–3 BACs using the procedure above, Velvet (Zerbino and Birney, 2008) is executed to assemble each BAC individually. As was done in (Lonardi *et al.*, 2013), we generated multiple assembly for several choices of Velvet's l-mer (hash) size (25–79, step of 6). The assembly reported is the one that maximizes the n50 (n50 indicates the length for which the set of all contigs of that length or longer contains at least half of the total size of all contigs).

We employed several metrics to evaluate the improvement in read decoding and assembly enabled by the slicing algorithm. For one of the barley sets (Hv10) we executed HashFilter using several choices of k ($k = 20, 23, 26, 29, 32$) on the full 748 M reads dataset (i.e. with no slicing) as well as with $k = 32$ using the slicing algorithm described ealrier. The first five rows of Table 1 summarize the decoding results. First, observe that as we increase k, the number of decoded reads increases monotonically. However, if one fixes k (in this case $k = 32$, which is the maximum allowed by HashFilter), slicing Hv10 in 4 slices of ≈4 M reads increases significantly the number of decoded reads (84.60 compared with 77.19%) available for assembly. Analysis of the number of assignments to ghost BACs also shows significant improvement in the decoding accuracy when using slicing: 0.000086% of the reads are assigned to unused BAC signatures compared with 0.000305–0.001351% when HashFilter is used on the full dataset. We carried out a similar analysis on

Table 1. Decoding and assembly statistics for the Hv10 barley set for several choices of k on the full dataset, and for the improved slicing algorithm

	No slicing					Slicing
	$k=20$	$k=23$	$k=26$	$k=29$	$k=32$	$k=32$
Reads decoded (%)	67.76%	71.07%	73.57%	75.56%	77.19%	84.60%
Reads decoded (M)	511	536	555	570	582	617
Reads assigned to ghost BACs(%)	0.000498%	0.000305%	0.000480%	0.000484%	0.001351%	0.000086%
Reads to be assembled (M)	704	724	739	748	723	695
Coverage (×)	502	502	528	502	517	499
Reads used by VELVET (%)	73.6%	77.9%	80.8%	81.8%	86.7%	90.7%
n50 (bp)	3 634	5 143	7 069	8 877	12 260	42 819
Sum/size (%)	102.8%	102.8%	100.5%	97.9%	89.5%	121.9%
Observed genes (27 expected)	20	20	20	20	20	20
Coverage of observed genes (%)	94.0%	94.0%	94.0%	94.0%	94.1%	94.0%

Table 2. A subset of 26 BACs in Hv10 have a 454-based assembly available from (Stein *et al.*, 2012)

	No slicing $k=32$	Slicing $k=32$
0 mismatches	75.2%	82.4%
1 mismatch	78.7%	85.9%
2 mismatches	80.5%	87.4%
3 mismatches	82.3%	88.7%

The table reports the percentage of the reads for those 26 BACs that can be mapped (with BOWTIE with 0, 1, 2 and 3 mismatches) to the corresponding assemblies.

Hv9: when the full dataset was processed with HASHFILTER ($k=26$), the number of reads assigned to ghost BACs was very high, ≈1.9 M reads out of 196 M (0.9653%). When the optimal slicing is used ($k=32$), only 19 140 reads out of 516 M are assigned to ghost BACs (0.0037%). Also, observe in Table 1 how the improved decoding affects the quality of the assembly for Hv10. When comparing no slicing to slicing-based decoding, the average n50 jumps from 12 260 to 42 819 bp (both for $k=32$) and the number of reads used by VELVET in the assembly increases from 86.7 to 90.7%.

For Hv10, we also measured the number of decoded reads that map (with 0, 1, 2 and 3 mismatches) to the assembly of a subset of 26 BACs that are available from (Stein *et al.*, 2012). Table 2 reports the average percentage of decoded reads (either from the full dataset or from the optimal slicing) that BOWTIE can map to the 454-based assemblies. Observe how the slicing step improves by 6–7% the number of reads mapped to the corresponding BAC assembly, suggesting a similar improvement in decoding accuracy. Similar improvements in decoding accuracy were observed on the other datasets (data not shown).

On Hv8, we investigated the effect of the slice size on the decoding and assembly statistics: earlier we claimed that the optimal size corresponds to the peak of the graphs in Figure 2. For instance, notice that the peak for Hv8 is ≈2 M reads. We decoded and assembled reads using slicing sizes of 2 M reads as well as (non-optimal) slice size of 3 M reads. The experimental results are shown in Table 3. Observe that the decoding with 3 M does not achieve the same decoding accuracy or assembly quality of the slicing with 2 M, but again both are significantly better than without slicing. Again, notice in Table 3 how improving the read decoding affects the quality of the assembly. The average n50 increases from 4126 bp ($k=26$, no slicing) to 34 262 bp ($k=32$, optimal slicing) and the number of reads used by VELVET in the assembly increases from 55.6 to 91.2%, respectively. For Hv8, 207 genes were known to belong

Table 3. Decoding and assembly statistics for Hv8: comparing no slicing and slicing with two different slice sizes (2 M reads is optimal according to the peak in Fig. 2)

	No slicing	Slicing	
	$k=26$	$k=32$, 3 M	$k=32$, 2 M
Reads decoded (%)	31.68%	78.98%	82.74%
Reads decoded (M)	270	539	600
Reads to be assembled (M)	289	591	669
Coverage (x)	94	197	223
Reads used by VELVET (%)	69.0%	92.6%	91.6%
n50 (bp)	4126	31 226	34 262
Sum/size (%)	55.6%	97.0%	102.0%
Observed genes (207 expected)	178	190	187
Coverage of observed genes (%)	86.0%	91.1%	91.2%

to a specific BAC clone (Lonardi *et al.*, 2013): the assembly using slicing-based coding recovered at least 50% of the sequence of 187–190 of them, compared with 178 using no slicing.

Finally, we compared the performance of our slicing method against the experimental results in (Lonardi *et al.*, 2013), which were obtained by running HASHFILTER with no data slicing ($k=26$). The basic decoding and assembly statistics when no slicing is used are reported in Table 4. First, observe the large variability of results among the 10 sets. Although the average number of decoded reads for $k=26$ is ≈460 M, there are sets which have less than half that amount (Hv6 and Hv9) and sets have more than twice the average (e.g. Hv3). As a consequence, the average fold-coverage ranges from 72 × (Hv6) to 528 × (Hv10). In general, the assembly statistics (without slicing-based decoding) are not very satisfactory: the n50 ranges from 2630 (Hv9) to 8190 bp (Hv3); the percentage of reads used by VELVET ranges from 66.0 (Hv9) to 85.9% (Hv3 and Hv4); the percentage of known genes covered at least 50% of their length by the assemblies ranged from 66% (Hv4) to 97% (Hv3).

When we decoded the same 10 datasets using the optimal slice size (using this time $k=32$) the assemblies improved drastically. The decoding and assembly statistics are summarized in Table 5: note that each set has its optimal size and the corresponding number of slices. First observe how the number of decoded reads increased significantly for most datasets (e.g. 330–785 M for Hv7, 289–669 M for Hv8, 209–516 M for Hv9, 369–907 M for Vu1 and 448–695 M for Vu2). Only for two datasets the number of decoded reads decreased slightly (by 12 M reads in Hv5, and by 44 M in Hv10). For all the datasets, the average n50 increased significantly—from an average of about 5.7 to 30 kbp (see Supplementary Dataset 2 for detailed assembly statistics on each dataset). Even for datasets for

Table 4. Decoding and assembly statistics for the 10 datasets using $k = 26$ on the full dataset (no slicing)

	Decoding		*Velvet assembly* ($l = 25, 31, \ldots 79$, best n50)			
	Reads (M)	*Coverage* (×)	*n50* (bp)	*Reads used* (%)	*Sum/size* (%)	*Observed/expected genes* (%)
Hv3	1099.0	431.0	8190	85.9	96.7	1 433/1 471$_{(97.42)}$
Hv4	393.2	135.5	5718	85.9	85.9	312/473$_{(65.96)}$
Hv5	483.0	158.9	8048	84.5	93.2	194/226$_{(85.84)}$
Hv6	218.0	72.0	6032	83.2	79.9	208/244$_{(85.25)}$
Hv7	330.0	110.0	5352	75.9	63.7	201/228$_{(88.16)}$
Hv8	289.0	94.2	4126	69.0	55.6	178/207$_{(85.99)}$
Hv9	208.0	95.8	2630	66.0	38.5	262/361$_{(72.58)}$
Hv10	739.0	528.0	7069	80.8	100.5	20/27$_{(74.07)}$
Vu1	369.0	88.5	4150	67.6	49.5	461/612$_{(75.33)}$
Vu2	448.0	126.0	5670	75.1	56.1	406/503$_{(80.72)}$
Average	457.6	184.0	5699	77.4	72.0	(81.13%)

Table 5. Decoding and assembly statistics for the 10 datasets using $k = 32$ and optimal slicing

Slicing	*Decoding*		*Velvet assembly* ($l = 25, 31, \ldots 79$, best n50)			
(No. Slices × size)	*Reads* (M)	*Coverage* (×)	*n50* (bp)	*Reads used* (%)	*Sum/size* (%)	*Observed/expected genes* (%)
Hv3 (11 × 2 M)	1156.0	460.0	28 477	90.4	123.0	1437/1471$_{(97.69\%)}$
Hv4 (8 × 2 M)	595.6	205.9	28 341	93.9	114.5	319/473$_{(67.44\%)}$
Hv5 (4 × 4 M)	471.0	155.5	31 038	93.6	101.0	196/226$_{(86.73\%)}$
Hv6 (6 × 1.5 M)	243.0	81.1	25 194	92.9	89.4	206/244$_{(84.43\%)}$
Hv7 (15 × 3 M)	785.0	264.0	39 742	91.1	104.0	204/228$_{(89.47\%)}$
Hv8 (12 × 2 M)	669.0	223.0	34 262	91.6	102.0	187/207$_{(90.34\%)}$
Hv9 (14 × 1.25 M)	516.0	246.0	32 634	94.3	103.2	309/361$_{(85.60\%)}$
Hv10 (4 × 5 M)	695.0	499.0	42 819	90.7	121.9	20/27$_{(74.07\%)}$
Vu1 (12 × 1.5 M)	907.0	232.0	16 388	89.7	89.7	510/612$_{(83.33\%)}$
Vu2 (14×1.5 M)	970.0	283.0	20 748	91.5	93.6	446/503$_{(88.67\%)}$
Average	700.8	265.0	29 964	92.0	104.2	(84.78%)

which slicing decreased the number reads (Hv5 and Hv10), the n50 increased significantly. The number of reads used by VELVET increased from an average of 77–92%; the fraction of known genes that were recovered by the assemblies increased from 81 to 85%. We recognize that the improvement from Tables 4 to 5 is not just due to the slicing, but also to the increased k (from 26 to 32). We have already addressed this point in Tables 1–3, where we showed that increasing k from 26 to 32 helps the decoding/assembly but the main boost in accuracy and quality is due to slicing. Recall that the assemblies in Tables 4 and 5 were carried out using VELVET with $l = 25, 31, \ldots, 79$ and choosing the assembly with the largest n50. On the Hv3 dataset, we have also tested VELVET with fixed $l = 49$, SPADES (Bankevich *et al.*, 2012) with $l = 31, 33, \ldots, 79$, and IDBA-UD (Peng *et al.*, 2012) with $l = 31, 33, \ldots, 79$ (see Supplementary Table S3). VELVET (best n50) and SPADES' performance were comparable, while IDBA-ud achieved lower n50. We also tested VELVET with $l = 49$, and SPADES with $l = 31, 33, \ldots, 79$ on all the other datasets (Supplementary Table S3). Setting $l = 49$ for VELVET led to less 'bloated' assemblies, somewhat comparable to SPADES' output.

As a final step, we investigated how the depth of sequencing affects BAC assembly quality. To this end, we multiplexed 16 barley BACs on one lane of the Illumina HiSeq2000, using custom multiplexing adapters. The size of these BACs ranged ≈70–185 kbp (see Supplementary Table S2). After demultiplexing the sequenced reads, we obtained 34.4 M 92-bases paired-end reads (insert size of 275 bases). We quality-trimmed the reads, then cleaned them of spurious sequencing adaptors; finally reads affected by *E.coli* contamination or BAC vector were discarded. The final number of cleaned reads was 23.1 M, with an average length of ≈88 bases. The depth of sequencing for the 16 BACS ranged from ≈6600× to 27 700× (see Supplementary Table S2).

Another set of 52 barley BACs was sequenced by the Department of Energy Joint Genome Institute using Sanger long reads. All BACs were sequenced and finished using PHRED/PHRAP/CONSED to a targeted depth of 10×. The primary DNA sequences for each of these 52 BACs were assembled in one contig, although two of them were considered partial sequence.

The intersection between the set of 16 BACs sequenced using the Illumina instrument and the set of 52 BACs sequenced using Sanger is a set of seven BACs (highlighted in bold in Supplementary Table S2), but one of these seven BACs is not full-length (052L22). We used the six full-length Sanger-based BAC assemblies as the 'ground truth' to assess the quality of the assemblies from Illumina read at increasing depth of sequencing. To this end, we generated datasets corresponding to 100, 250, 500, 1000, 2000, 3500, 5000, 6000, 7000 and 8000× depth of sequencing (for each of the six BACs), by sampling uniformly short reads from the high-depth datasets. For each choice of the depth of sequencing, we generated 20 different datasets, for a total of 1200 datasets. We assembled the reads on each dataset with VELVET v1.2.09 (with hash value $k = 79$ to minimize the probability of false overlaps) and collected statistics for the resulting assemblies. Figure 3 shows the value of n50 (A), the size of

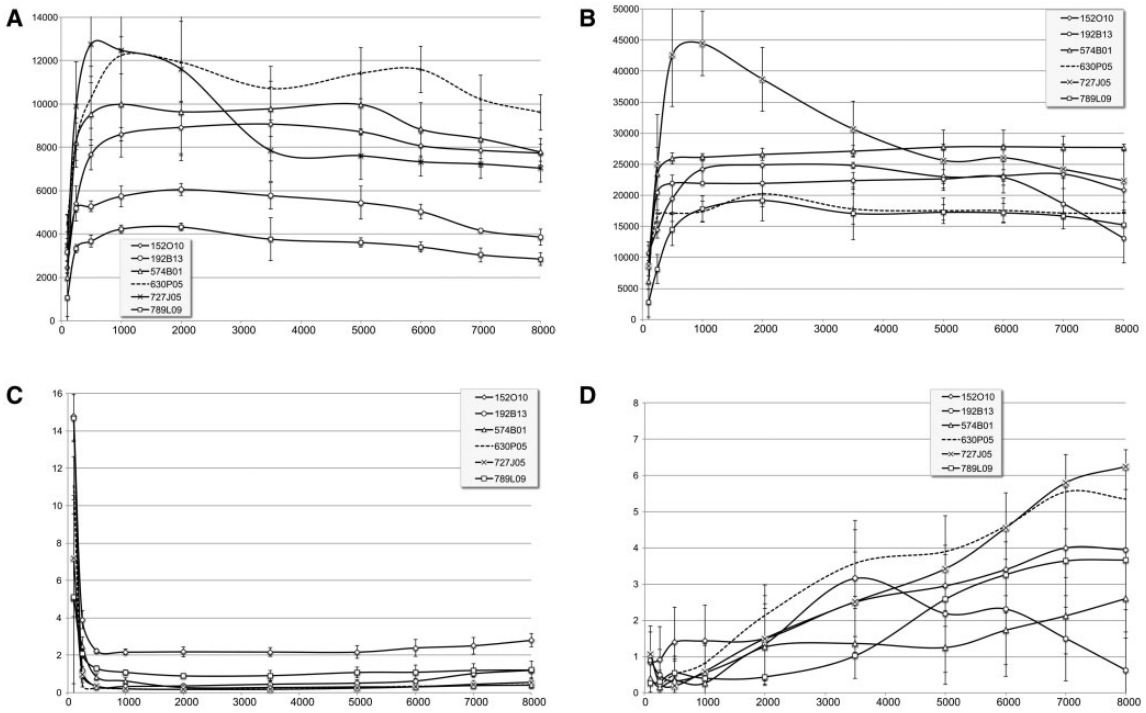

Fig. 3 VELVET assembly statistics as a function of the depth of sequencing coverage: **(A)** n50, **(B)** longest contig, **(C)** percentage of the target BAC not covered by the assembly, **(D)** number of assembly errors; each point is an average over 20 samples of the reads, errors bars indicate standard deviation among the samples

the largest contig (B), the percentage of the target BAC not available in the assembly (C) and number of assembly errors (D) for increasing depth of sequencing. Each point in the graph is the average over the 20 datasets, and error bars indicate the SD. In order to compute the number of assembly errors we used the tool developed for the GAGE competition (Salzberg *et al.*, 2011). According to GAGE, the number of assembly errors is defined as the number of locations with insertion/deletions of at least six nucleotides, plus the number of translocations and inversions.

A few observations on Figure 3 are in order. First, note that both the n50 and the size of the longest contig reach a maximum in the 500–2000 × range, depending on the BAC. Also observe that in order to minimize the percentage of BAC missed by the assembly one needs to keep the depth of sequencing below 2500 × (too much depth decreases the coverage of the target). Finally, it is very clear from (D) that as the depth of sequencing increases so do the number of assembly errors (with the exception of one BAC).

We have also investigated whether similar observations could be drawn for other assemblers. In Figure 4, we report the same assembly statistics, namely (A) the value of n50, (B) the size of the largest contig, (C) the percentage of the target BAC not available in the assembly and (D) number of assembly errors for increasing depth of sequencing for one of the BACs. This time we used three assemblers, namely VELVET, SPADES v3.1.1 (Bankevich *et al.*, 2012) and IDBA-UD (Peng *et al.*, 2012) (statistics for all BACs are available in Supplementary Figs S3–6). Although there are performance differences among the three assemblers, the common trend is that as the coverage increases, the n50 and the size of the largest contig decreases, while the percentage of the BAC missing and the number of assembly errors increases. Among the three assemblers, SPADES appears to be less affected by high coverage. SPADES was run with hash values $k = 25, 45, 65$ and option `careful` (other parameters were default). IDBA-UD was run with hash values $k = 25, 45, 65$ (other parameters were default). The reported assembly is the one chosen by IDBA-UD.

Independently from us, the authors of (Desai *et al.*, 2013) made similar observations on assembly degratadation. In their study, the authors assembled *E.coli* (4.6 MB), *Saccharomyces kudriavzevii* (11.18 MB) and *Caenorhabditis. elegans* (100 MB) using SOAPDENOVO, VELVET, ABYSS, MERACULOUS and IDBA-UD at increasing sequencing depths up to 200×. Their analysis showed that the optimum-sequencing depth for assembling these genomes is about 100×, depending on the specific genome and assembler.

Finally, we analyzed the performance of IDBA-UD, SPADES and VELVET on simulated reads. We generated 100 bp × 2 paired-end reads from the Sanger assembly of BAC 574B01 using the read simulator WGSIM (github.com/lh3/wgsim) at 100, 250, 500, 1000, 2000, 3500, 5000, 6000, 7000 and 8000 × depth of sequencing. Insert length was 250 bp, with a standard deviation of 10 bp. For each depth of sequencing, we generated simulated reads at 0, 0.5, 1 and 2% sequencing error rate (substitutions). Insertions and deletions were not allowed.

IDBA-UD was executed with hash values $k = 25, 45, 65$ (other parameters were default). VELVET was run with $k = 49$. We repeated the simulations 20 times for IDBA-UD and 10 times for VELVET and SPADES. In Supplementary Figures S7–9, we report the usual assembly statistics, namely n50, largest contig, percentage missing, and number of assembly errors for VELVET, IDBA-UD and SPADES on these datasets. Observe that with 'perfect' reads (0% error rate), ultra-deep coverage does not affect the performance of IDBA-UD and VELVET. With higher and higher sequencing errors, however, similar behaviors to the assembly of real data can be observed for IDBA-UD and VELVET: n50 and longest contig rapidly decrease, and missing portions of the BAC and number of mis-assemblies increase. Surprisingly, SPADES seems to be immune to higher sequencing error rates.

4 Discussion

Because the introduction of DNA sequencing in the 70s, scientists had to come up with clever solutions to deal with the problem of

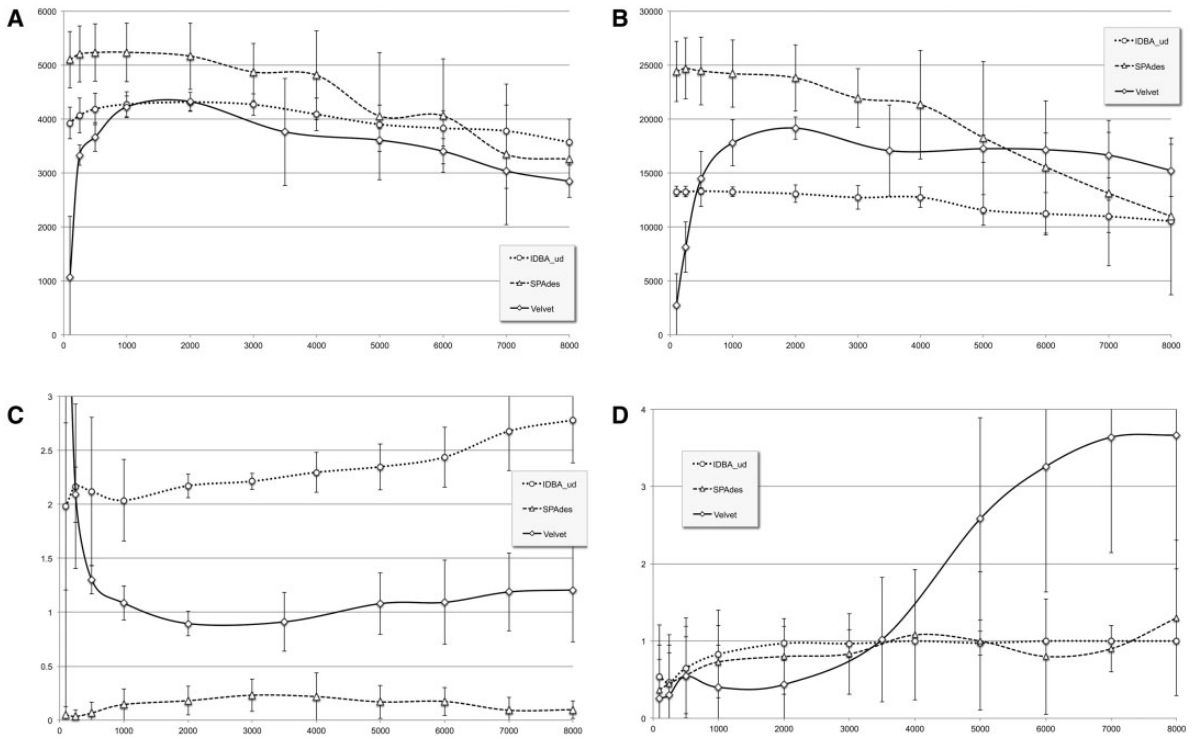

Fig. 4. Assembly statistics as a function of the depth of sequencing coverage for BAC 789L09 for three assemblers: VELVET, SPADES and IDBA; **(A)** n50, **(B)** longest contig, **(C)** percentage of the target BAC not covered by the assembly, **(D)** number of assembly errors; each point is an average over 10 subsamples of the reads, errors bars indicate standard deviation among the samples

de novo genome assembly with limited depth of sequencing. As the cost of sequencing keeps decreasing, one can expect that computational biologists will have to deal with the opposite problem: excessive amount of sequencing data. The Lander-Waterman-Roach theory (Lander and Waterman, 1988; Roach, 1995) has been the theoretical foundation to estimate gap and contig lengths as a function of the depth of sequencing. We do not have a theory that would explain why the quality of the assembly starts degrading when the depth is too high. Possible factors include the presence (in real data) of chimeric reads, sequencing errors, and read duplications, or their combination thereof.

In this study, we report on the *de novo* assembly of BAC clones, which are relatively short DNA fragments (100–150 kbp). With current sequencing technology it is very easy to reach depth of sequencing in the range of 1000–10 000× and study how the assembly quality changes as the amount of sequencing data increases. Our experiments show that when the depth of sequencing exceeds a threshold the overall quality of the assembly starts degrading (Fig. 3). This appears to be a common problem for several *de novo* assemblers (Fig. 4). The same behavior is observed for the problem of decoding reads to their source BAC (Fig. 2), which is the main focus of this article.

The important question is how to deal with the problem of excessive sequencing depth. For the decoding problem we have presented an effective 'divide and conquer' solution: we 'slice' the data in subsamples, decode each slice independently, then merge the results. In order to handle conflicts in the BAC assignments (i.e. reads that appear in multiple slices that are decoded to different sets of BACs), we devised a simple set of voting rules. The question that is still open is what to do for the assembly problem: one could assemble slices of the data independently, but it is not clear how to merge the resulting assemblies. In general, we believe that the problem of *de novo* sequence assembly must be revisited from the ground up under the assumption of ultra-deep coverage.

Acknowledgements

We thank the Department of Energy Joint Genome Institute (Dr Jane Grimwood and Dr Jeremy Schmutz) for the use of the reference BAC barley clone data 14090–14118 assembled from Sanger sequencing data. We also thank Prof Titus Brown (Michigan State U.) and Prof. Pavel Pevzner (UC San Diego) for useful comments on this study; Dr Ming-Cheng Luo (UC Davis) for fingerprinting barley and cowpea BACs.

Funding

This work was supported in part by the US National Science Foundation (DBI-1062301) and (IIS-1302134), by the USDA National Institute of Food and Agriculture (2009-65300-05645), by the USAID Feed the Future program (AID-OAA-A-13-00070) and the UC Riverside Agricultural Experiment Station Hatch Project CA-R-BPS-5306-H.

Conflict of Interest: none declared.

References

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

Bozdag,S. *et al.* (2007) A compartmentalized approach to the assembly of physical maps. In: *Proceedings of IEEE International Symposium on Bioinformatics & Bioengineering (BIBE'07)*, Boston, MA, pp. 218–225.

Bozdag,S. *et al.* (2013) A graph-theoretical approach to the selection of the minimum tiling path from a physical map. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, **10**, 352–360.

Desai,A. *et al.* (2013) Identification of optimum sequencing depth especially for *de novo* genome assembly of small genomes using next generation sequencing data. *PLoS One*, **8**, e60204.

Ding,Y. *et al.* (2001) Five-color-based high-information-content fingerprinting of bacterial artificial chromosome clones using type IIS restriction endonucleases. *Genomics*, **74**, 142–154.

Lander,E.S. and Waterman,M.S. (1988) Genomic mapping by fingerprinting random clones: a mathematical analysis. *Genomics*, **2**, 231–239.

Lonardi,S. *et al.* (2013) Combinatorial pooling enables selective sequencing of the barley gene space. *PLoS Comput. Biol.*, **9**, e1003010.

Luo,M.-C. *et al.* (2003) High-throughput fingerprinting of bacterial artificial chromosomes using the snapshot labeling kit and sizing of restriction fragments by capillary electrophoresis. *Genomics*, **82**, 378–389.

Muñoz-Amatriaín,M. *et al.* (2015) Sequencing of 15 622 gene-bearing BACs reveals new features of the barley genome. *bioRxiv, doi:10.1101/018978*.

Peng,Y. *et al.* (2012) IDBA-UD: a *de novo* assembler for single-cell and metagenomic sequencing data with highly uneven depth. *Bioinformatics*, **28**, 1420–1428.

Roach,J.C. (1995) Random subcloning. *Genome Res.*, **5**, 464–473.

Salzberg,S.L. *et al.* (2011) GAGE: a critical evaluation of genome assemblies and assembly algorithms. *Genome Res.*, **22**, 557–567.

Soderlund,C. *et al.* (2000) Contigs built with fingerprints, markers, and FPC v4.7. *Genome Res.*, **10**, 1772–1787.

Stein,N. *et al.* (2012) A physical, genetic and functional sequence assembly of the barley genome. *Nature*, **491**, 711–716.

Thierry-Mieg,N. (2006) A new pooling strategy for high-throughput screening: the shifted transversal design. *BMC Bioinformatics*, **7**, 28.

Yu,Y. *et al.* (2000) A bacterial artificial chromosome library for barley (*Hordeum vulgare L.*) and the identification of clones containing putative resistance genes. *Theor. Appl. Genet.*, **101**, 1093–1099.

Zerbino,D. and Birney,E. (2008) VELVET: Algorithms for *de novo* short read assembly using de Bruijn graphs. *Genome Res.*, **8**, 821–829.