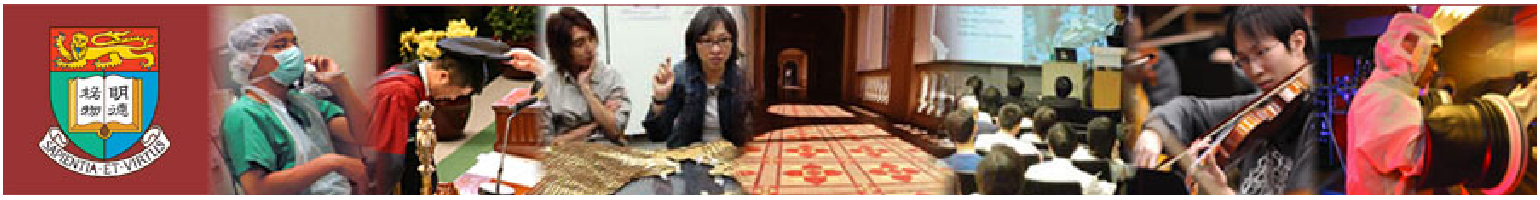

Category

IDBA-UD: A de Novo Assembler for Single-Cell and Metagenomic Sequencing Data with Highly Uneven Depth

Yu Peng[1], Henry C.M. Leung[1], S.M. Yiu[1] and Francis Y.L. Chin[1,*]

[1]Department of Computer Science, The University of Hong Kong
Pokfulam Road, Hong Kong

Received on XXXXX; revised on XXXXX; accepted on XXXXX

Associate Editor: XXXXXXX

ABSTRACT

Motivation: Next-generation sequencing allows us to sequence reads from a microbial environment using single cell sequencing or metagenomic sequencing technologies. However, both technologies suffer from the problem that sequencing depth of different regions of a genome or genomes from different species are highly uneven. Most existing genome assemblers usually have an assumption that sequencing depths are even. These assemblers fail to construct correct long contigs.

Results: We introduce the IDBA-UD algorithm that based on the de Bruijn graph approach for assembling reads from single cell sequencing or metagenomic sequencing technologies with uneven sequencing depths. Several non-trivial techniques have been employed to tackle the problems. Instead of using a simple threshold, we use multiple depth relative thresholds to remove erroneous *k*-mers in both low-depth and high-depth regions. The technique of local assembly with paired-end information is used to solve the branch problem of low-depth short repeat regions. To speed up the process, an error correction step is conducted to correct reads of high-depth regions that can be aligned to high confident contigs. Comparison of the performances of IDBA-UD and existing assemblers (Velvet, Velvet-SC, SOAPdenovo and Meta-IDBA) for different datasets shows that IDBA-UD can reconstruct longer contigs with higher accuracy.

Availability: The IDBA-UD toolkit is available at our website http://www.cs.hku.hk/~alse/idba_ud.

1 INTRODUCTION

Because over 99% of microbes cannot be cultivated, single cell sequencing and metagenomic sequencing technologies are used to study these microbes (Chitsaz, et al., 2011; Wooley, et al., 2010). Single cell sequencing technology amplifies and sequences genome of an individual cell without cultivation (Chitsaz, et al., 2011). Because of the amplification bias, the sequencing depths at different regions of the genome can be extremely uneven. Metagenomic sequencing studies a microbe community as a whole (Wooley, et al., 2010) and has similar problem of uneven sequencing depths of genomes because different species in a sample have different abundances. Almost all existing de novo assembly tools were designed for single genome with uniform sequencing depth and were used by some recent studies on microbes (Rodrigue, et al., 2009; Woyke, et al., 2009). However, these tools may not be able to produce long contigs when applying to data with highly uneven sequencing depths.

Many existing de novo assembly tools for the next generation sequencing (NGS) reads adopt the de Bruijn graph approach (Butler, et al., 2008; Chaisson, et al., 2009; Li, et al., 2010; Peng, et al., 2010; Pevzner, et al., 2001; Simpson, et al., 2009; Zerbino and Birney, 2008) in which a vertex represents a unique length-*k* substring called *k*-mer and an edge connects vertices *u* and *v* if and only if *u* and *v* appear consecutively in a read. Each read is represented by a path of *k*-mers in the de Bruijn graph. After error detection and removal, a simple path in the de Bruijn graph represents a contig.

There are three major problems in this approach (Peng, et al., 2010):

a) Incorrect *k*-mers: Sequencing errors introduce many incorrect *k*-mers (vertices) that make the de Bruijn graph complicated.

b) Gap problem: When *k* is large, especially in regions with lower sequencing depths, some *k*-mers (i.e., vertices, also edges, in the de Bruijn graph) are missing.

c) Branching problem: Due to repeat regions or erroneous reads, many branches are introduced in the de Bruijn graph especially when *k* is small.

For Problem (a), some of these errors can be removed by the topological structure of the graph. For the remaining errors, based on the assumption of uniform sequencing depth and the observation that the multiplicity of an erroneous *k*-mer is usually smaller than that of a correct *k*-mer, existing tools use a simple threshold to either prune contigs if the contigs are formed by *k*-mers of low multiplicity (e.g., Velvet (Zerbino and Birney, 2008; Zerbino, et al., 2009) and Abyss (Simpson, et al., 2009)) or directly remove *k*-mers with low multiplicity (IDBA(Peng, et al., 2010) and EULER-SR(Chaisson and Pevzner, 2008)). Note that this also solves some of the branching problems (Problem (c)) due to incorrect *k*-mers.
For Problems (b) and (c), using a small *k* will induce more branches while using a large *k* will result in more gaps. Most existing tools (e.g. Velvet (Zerbino and Birney, 2008) and SOAPdenovo (Li, et al., 2010)) just pick an appropriate *k*, some intermediate value, to balance the two problems. On the other hand, the IDBA assembler (Peng, et al., 2010) provides a better solution which, instead of

[*]To whom correspondence should be addressed.

using a single k, iterates from $k = k_{\min}$ to $k = k_{\max}$. At each iteration, the constructed contigs are used as reads for the next iteration. These contigs carry the k-mers of the current iteration, which may be missing in the next iteration, to the next iteration, thus solving some of the gap problems. It then relies on larger k to resolve the branches for the repeat regions.

However, when applying to single cell or metagenomic assembling, highly uneven sequencing depth aggravates these problems further that affect the performance of these tools substantially due to the following issues. Issue (A): erroneous vertices and branches in high-depth regions; Issue (B): gaps in low-depth short repeat regions.

Problems (a) and (c) due to Issue (A):
Due to highly uneven sequencing depth, the assumption of an incorrect k-mer having lower multiplicity is not valid. Those incorrect ones in the high-depth regions may even have higher multiplicity than the correct ones in the low-depth regions, thus simply using a single threshold to remove incorrect vertices will not work. Setting the threshold too low induces many incorrect vertices and edges (those in high-depth regions) in the graph. Setting the threshold too high will remove many correct vertices and edges in low-depth regions. We remark that there exist some error correction algorithms for reads/k-mers (Chaisson and Pevzner, 2008; Kelley, et al., 2010; Medvedev, et al., 2011), but they do not perform very well in data sets with very uneven sequencing depths.

Problems (b) and (c) due to Issue (B):
Recall that most existing assemblers do not have a good method to resolve Problems (b) and (c) probably except IDBA. Even for IDBA, in low-depth short repeat regions[1], when k is small, the branching problem makes it difficult to construct a contig to be passed to the next iteration. When k is increased, due to the low-depth issue, we still have the missing k-mer problem (the gap problem).

Velvet-SC (Chitsaz, et al., 2011) is the only tool that tries to address the assembling problem of single cell sequencing data with very uneven sequencing depths. Following Velvet, Velvet-SC picks an appropriate k to balance the gap and the branching problem; and uses variable thresholds to address problems related to Issue (A). Short erroneous contigs are filtered iteratively using different thresholds from low to high sequencing depths based on a global average of the multiplicity of all k-mers. Its performance is already better than existing tools designed for even sequencing depth. However, problems related to Issue (B) are not yet handled. In this paper, we propose an assembler called IDBA-UD for de novo assembly of reads with uneven sequencing depths that tackles both issues.

To resolve Issue (A), IDBA-UD extends and enhances the idea of variable thresholds of Velvet-SC (Chitsaz, et al., 2011) to filter out erroneous contigs. In order to cater for very extreme sequencing depths, instead of using a global average of the multiplicity of all k-mers, we adopt variable *relative* thresholds depending on the sequencing depths of their neighboring contigs based on the idea that short contigs with much lower sequencing depths than their neighboring contigs tend to be erroneous. For the gap and branching problems, we follow the approach of IDBA and iterate from a small k to a large k so that the missing k-mers for large k can be obtained from contigs constructed in the iterations of small k.

Then we tackle Issue (B) as follows. The problem of Issue (B) is due to the low-depth short repeat regions such that using small k, we cannot get the contig out since it is a repeat region and the branches may be complicated due to the ambiguity of using a small value of k. When k increases, however, due to the low sequencing depths some k-mers are missing. Even if we iterate from small k to large k, this problem of missing k-mers cannot be resolved. So, we employ the technique of local assembly with paired-end information to handle these cases. Paired-end reads with one end aligned to some long confident contigs are grouped together. Local assembly is performed on the unaligned ends. Since we consider only the read pairs with one end aligned to the contig, the ambiguity due to small k is removed. If the insert size is longer than the repeat involved, it is likely that we can extend the contig over this repeat region, thus constructing the missing k-mers for large k. Note that this local assembly step can also help to resolve some branching problems in high-depth regions too.

To further reduce the size of the de Bruijn graph and to speed up the assembly process, at every iteration, we conduct an additional error correction step by aligning the erroneous reads from the high-depth regions to confident contigs (i.e., with many supporting reads) which turns out to be very effective.

We compared the performance of IDBA-UD with other assemblers on data in actual situations when the sequencing depths are extremely uneven, e.g., with the ratios larger than 100 : 1. Experiments on both simulated and real data sets showed that IDBA-UD produces much longer contigs than existing assemblers with higher coverage and precision.

2 METHODS

A flowchart of the major steps of IDBA-UD is shown in Figure 1. IDBA-UD iterates the value of k from $k_{\min}$ to $k_{\max}$. In each iteration, an *accumulated de Bruijn graph* H_k for a fixed k is constructed from the set of input reads and the contigs ($C_{k\text{-s}}$ and $LC_{k\text{-s}}$) constructed in previous iterations, i.e. these contigs are treated as input reads for constructing H_k. In each iteration, IDBA-UD also progressively increases the value of depth cutoff thresholds for removing some low-depth contigs so as to get longer confident contigs (C_k) in H_k. Error in reads are corrected by aligning the reads to some confident contigs. Some missing k-mers in reads can be recovered from those contigs (LC_k) reconstructed by local assembling a small set of paired-end reads with one end aligned to a confident contig. Information of these missing k-mers will be passed to the next iteration through these contigs (LC_k) for the construction of H_{k+s}. Finally, all outputted contigs are used to form scaffolds using paired-end reads information.

Algorithm 1 shows the pseudocode of IDBA-UD for assembling a set of paired-end reads R with insert distance d and standard derivation δ. In the first iteration when $k = k_{\min}$, H_k is equivalent to a de Bruijn graph for vertices whose corresponding k-mers have multiplicity at least m (2 by default) times in all reads. During all the subsequent iterations, some sequencing errors are first removed according to the topological structure of H_k, e.g., dead-end contigs

[1] For very long repeats (longer than the whole span of a paired-end read), it is almost impossible to resolve it.

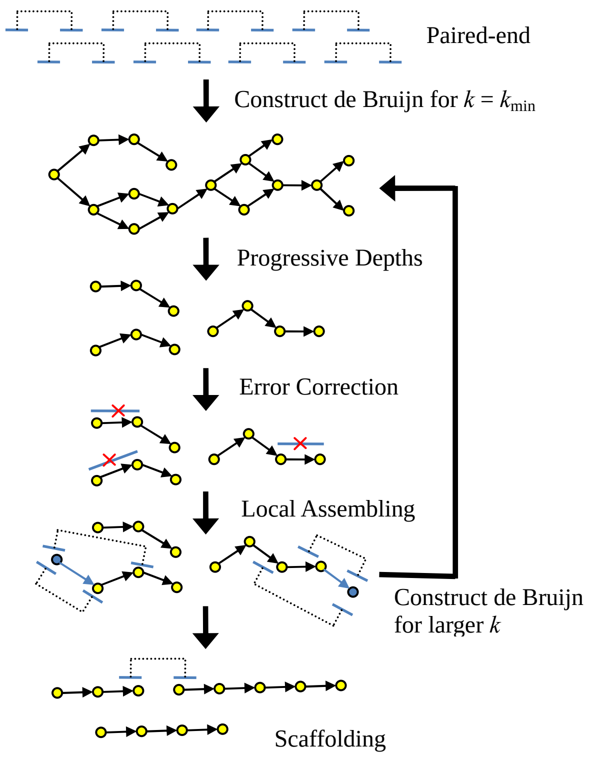
Paired-end
Construct de Bruijn for $k = k_{\min}$
Progressive Depths
Error Correction
Local Assembling
Construct de Bruijn for larger k
Scaffolding

Fig. 1. Flowchart of IDBA-UD.

and bubbles (Steps (b) and (c)). The dead-end contigs (tangling paths in H_k of lengths shorter than $2k$) are likely to be false positives (Li, et al., 2010; Simpson, et al., 2009; Zerbino and Birney, 2008). Paths (bubbles) representing very similar contigs except at one position and with the same starting vertex and ending vertex are likely to be caused by an error or an SNP and they should be merged together into one contig (Hernandez, et al., 2008; Simpson, et al., 2009; Zerbino and Birney, 2008). When constructing H_{k+s} from H_k, each length s+1 path in H_k is converted into a vertex ((k+s)-mer) and there is an edge from between two vertices if the corresponding (k+s+1)-mer appears f (1 by default) times in reads or once in contigs in $C_k \cup LC_k$. In the following subsections, we will describe the other steps of IDBA-UD in details.

2.1 Progressive Relative Depth

The sequencing depth, *depth* in short, of each simple path (contig) in H_k (H'_k which is a copy of H_k is used in Algorithm 1 so as to preserve H_k after the implementation of this step) is used to remove errors. The *depth of a contig* is the average number of reads covering each k-mer in the contig. Note that long contigs are usually correct, because long simple paths can unlikely be formed by erroneous reads; similarly for high-depth contigs which have supports from many reads. For a contig, whether its length is long or short and whether its depth is high or low cannot be judged by its absolute values as the length of a contig depends on the value of k and the depth of a contig depends on the depths of its neighboring contigs (neighboring contigs can be identified by their adjacency in the de Bruijn graph). Even though wrong contigs in high-depth regions may have higher depths than correct contigs from low-depth regions, *short* (< 2k) and *relatively low-depth* (< a fraction β of its neighboring contigs' average depth) contigs are likely to be erroneous and can be removed. There is still a risk of removing short and relatively low-depth correct contigs, because some relatively low-depth correct contigs with high-depth neighbors may be broken into short contigs by some wrong contigs (as branches in H_k). Based on the observation that these short and relatively low-depth correct contigs usually have higher depths than the short wrong contigs, we can filter out these wrong contigs first by increasing the depth cutoff threshold progressively from low to high. After the wrong contigs or branches are removed by a low depth cutoff threshold, the relative low-depth correct contigs will be linked together to form long confident contigs which will be considered as reads for the next iteration.

The key idea to consider the depth progressively and relatively is shown in Algorithm 2. $T(c)$ represents the depth of contig c and $T_{neighbor}(c)$ represents the mean depth of c's neighboring contigs. The filtering depth cutoff threshold t is increased by a factor α progressively (α is about 10%). A geometric increase, instead of absolute increase (as used in Velvet-SC), in the depth cutoff threshold value improves implementation efficiency because the threshold difference is more sensitive at the low-depth values than the high-depth values. In each iteration, short contig c is removed if its depth $T(c)$ is lower than the minimum of cutoff threshold t and the relative threshold $\beta * T_{neighbor}(c)$ where β is in the range of 0.1 to 0.5.

Algorithm 1 IDBA-UD(R, d, δ) :

```
Pre-Error-Correction (optional)
Repeat from $k := k_{\min}$ to $k_{\max}$ with step $s$
    a)  If $k = k_{min}$, then construct $H_k$ from $R$
        else construct $H_k$ from $H_{k-s}$ and $R \cup C_{k-s} \cup LC_{k-s}$
    b)  Remove dead-ends with length < $2k$
    c)  Merge bubbles
    d)  $H'_k := H_k$
    e)  Progressive-Relative-Depth($H'_k$, $k$)
    f)  Get all potential contigs $C_k$ of $H'_k$
    g)  Error-Correction($C_k$, $R$)
    h)  $LC_k$ := Local-Assembly($C_k$ , $R$, $d$, $\delta$)
Construct scaffolds
return $C_{k_{max}}$ and scaffolds
```

Algorithm 2 Progressive-Relative-Depth(G, k) :

```
$t := 1$
repeat
    for each contig $c$ in $G$
        if $len(c) < 2k$ and $T(c) < min(t, \beta * T_{neighbor}(c))$
            remove $c$ from $G$
    $t := t * (1+\alpha)$
until t  $t > \max_{c \in G} T(c)$
```

2.2 Local Assembly

IDBA makes use of the contigs (containing the information of some missing k-mers for larger k) constructed in each iteration for the construction of the de Bruijn graphs of larger k. These missing k-mers may not exist in any of the reads but they might help to fill the gaps in the de Bruijn graphs for larger k. This approach still has a limitation that not all the missing k-mers, i.e., contigs containing these k-mers, can be constructed (so not all the gaps can be filled) because of branches. The main contribution of local assembly is to construct these contigs for the missing k-mers, especially in the low-depth regions, based on the information of paired-end reads to

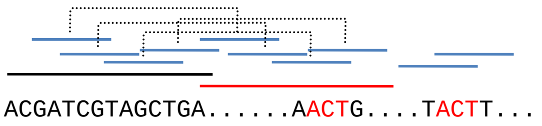

Fig. 2. Example of reconstructing missing *k*-mer in local assembly. ACT is a repeat region in the genome and no reads containing AACTG or TACTT for resolving repeat branches. In local assembly, ACT is no longer a repeat so that a simple path (local contig) covering AACTG can be reconstructed from local reads.

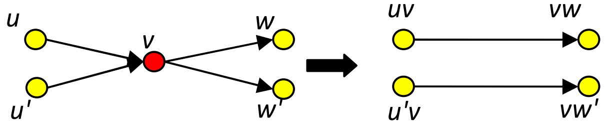

Fig. 3. Example of resolving repeats by iteration from k to k+1. The repeat region is a single *k*-mer, *uvw* and *u'vw'* appear in the genome. After the iteration, repeat *v* is resolved.

eliminate the branches introduced from other parts of the genome.

We shall illustrate this main idea of local assembly through an example (Figure 2). Let us consider the construction of a de Bruijn Graph for k=3, based on two reads, ...AACT and ACTG..., we have a simple path connecting the 3-mers, AAC, ACT and CTG. IDBA can reconstruct the missing 5-mer AACTG (not appeared in any reads) by forming a simple path containing it. However, as given in Figure 2, when ACT is a length-3 repeat in the genome (the repeat regions are apart by more than the insert distance) and there are reads covering the region ...TACTT... containing the other repeat. The 3-mer ACT in the de Bruijn graph for k=3 now has two in-branches and two out-branches (refer to the left diagram of Figure 3 where vertex v represents the 3-mer ACT; vertices u, w, u' and w' are for 3-mers AAC, CTG, TAC and CTT respectively). Under this situation, even when k is increased to 4 and 5 in IDBA (this part of graph will be disconnected in H_4 and H_5), the missing critical 5-mers AACTG cannot be reconstructed because of the branches. However, when considering the de Bruijn graph when k=3, IDBA-UD will align the paired-end reads to the contig ACGATCGTAGCTGA (Figure 2) while the reads of the other ends covering the repeat regions will only be ...AACT and ACTG... (reads covering the other repeat region ...TACTT... are not involved because they are far away). Thus local assembly (by considering the reads locally) can produce a simple path containing the critical 5-mer AACTG to resolve branches as if there were no repeats.

Algorithm 3 Local-Assembly(C, R, d, δ) :

```
Remove contigs shorter than $2l$ from $C$
Align reads in $R$ to contigs in $C$
For each $(r, s) \in R$
    if $r$ uniquely aligned to last $d+3\delta$ bases of $c$, $R_c := R_c \cup \{s\}$
    if $r$ uniquely aligned to last $d+3\delta$ bases of $c^{rc}$, $R_c^{rc} := R_c^{rc} \cup \{s\}$
For each $c \in C$
    $LC_c :=$ IDBA({last $d+3\delta$ bases of $c$} $\cup R_c$ )
    $LC_c^{rc} :=$ IDBA({last $d+3\delta$ bases of $c^{rc}$} $\cup R_c^{rc}$ )
return $\cup_{c \in G}(LC_c \cup LC_c^{rc})$
```

Let C_k be the set of contigs (simple paths) in H_k. The set of paired-end reads R_c are those with one read aligned with the ends of each long contig c (with length at least twice of read length) in C_k (c^{rc} stands for the reverse complement of contig c). The other unaligned ends of these aligned paired-end reads, which would cover the genome regions extended about an insert distance beyond each end of a long contig, are extracted separately. Assume the insert distances of paired-end reads satisfy the normal distribution $N(d, \delta)$. IDBA-UD groups the last $d + 3\delta$ bases of c/c^{rc} and R_c/R_c^{rc} together, and then locally assembles them into the set of local contigs LC_c/LC_c^{rc} using IDBA (Algorithm 1 without step e, g and h) as shown in Algorithm 3. Since those reads which are far away from the contig c will not be mixed up with these unaligned ends, the contig c and these unaligned ends (reads) of R_c can be used to construct a smaller and simpler de Bruijn graph whose simple paths (represented by the set of contigs LC_c) might reconstruct some of the missing k-mers and be considered as reads for the next iteration. Thus, the contigs can be extended longer and longer at each iteration. The expected number of resolved branches can be computed by Theorem 3 (Appendix).

2.3 Error-Correction

To reduce the errors in reads, error correction on some erroneous bases is performed based on the alignment between reads and confident contigs. Errors in reads are corrected only if they can be aligned to contigs with certain similarity, say 95%. The reads which can be multi-aligned to different contigs will not be considered for corrections. This approach of error correction is especially effective for high-depth regions because the confident contigs are well supported by many reads.

A position of a contig is labeled as "confirmed" if one base type appears over 80% in all reads aligned to that position. Each read, aligned to a contig region with all positions confirmed and the number of different bases no more than 3, will be corrected according to the confirmed bases.

A pre-error-correction step for improved efficiency can be used to remove errors in high-depth regions as the first step in IDBA-UD if the sequencing depths are extremely uneven. A medium k value and filtering threshold will be used to assemble reads to form contigs and errors in reads are corrected based on its alignment with the output contigs.

2.4 Scaffold

Last, reads are aligned to contigs so as to build a scaffold graph in which each vertex u represents a contig and each edge (u, v) represents the connection between u and v with a support of more than p (3 by default) paired-end reads. After the scaffold graph is built, scaffold algorithm (Li, et al., 2010) will be applied to further connect contigs.

3 RESULTS

In order to evaluate the performance of our algorithm, experiments[2] are carried out on several data sets with different properties. Results on existing general purpose assemblers like Velvet (Zerbino and Birney, 2008), SOAPdenovo (Li, et al., 2010), IDBA (Peng, et al., 2010) and special purpose assemblers like Velvet-SC (Chitsaz, et al., 2011), Meta-IDBA (Peng, et al., 2011) were compared. Different k values were tried for each assembler and the

[2] All experiments were done on a machine with 8-core 2.40GHz Intel CPU and 144GB memory. The tested assembler was run with multiple threads, if it supports.

result with best performance are shown and compared.

Two most important statistics N50 and coverage are calculated to evaluate the contiguity and completeness of assembly results. N50 is the length of the longest contig such that all the contigs longer than this contig cover at least half of the genome being assembled. (Earl, et al., 2011) Coverage is the proportion of the genome being covered by output contigs. In this paper, only correct contigs are considered in the calculation of N50 and coverage. A contig is considered as correct if it can be aligned to the genome reference by BLAT (Kent, 2002) with 95% similarity. For correct contigs, the substitution errors are computed by comparing the alignment between contigs and genome reference. For unaligned contigs, the number of contigs and number of bases are recorded for comparison.

# errors	# corrected	# TP	#FP
1856822	1627727	1626929 (99.95%)	798 (0.05%)

Table. 1 Error correction result on simulated 100x length-100 reads of *Lactobacillus delbrueckii* (~1.85Mb) with 1% error rate.

3.1.Error Correction

The performance of our error correction algorithm is assessed by correcting the simulated reads sampled from *Lactobacillus delbrueckii* genome (~1.85Mb). The simulated data set contains 1.85 million length-100 reads (100x) uniformly sampled from the reference with 1% error rate. The error correction algorithm was executed on this data set with output contigs of IDBA with k = 60 (k_{min} = k_{max} in this case). The correction result is shown in Table 1. There are 1856822 error bases in the data set. Our algorithm corrected 1627727 bases with 1626929 (99.95%) being true positive. Note that our target of error correction is to reduce the errors without introducing other errors. The remaining erroneous reads either contains too many errors to be aligned to contigs or are from those regions which cannot be assembled correctly. This high-precision and low-sensitivity error correction algorithm is suitable for IDBA-UD, because the remaining errors could be handled by other means in the later iterations. Note that error correction which simplifies the graph for next iterations improves efficiency tremendously.

3.2. Low depth assembly

Lactobacillus plantarum (~3.3Mb) was used as genome reference for simulating low-depth data set. 10x length-100 paired-end reads were simulated for testing. The assembly results of IDBA-UD, Velvet, SOAPdenovo and IDBA are shown in Table 2.

IDBA-UD has the longest N50 (36523), which is several folds of the N50 of Velvet, SOAPdenovo and IDBA (1584, 13761, and 8350). IDBA-UD also has the highest coverage (99.56%), which is higher than the coverage of Velvet, SOAPdenovo and IDBA (98.36%, 98.09% and 98.52%). IDBA-UD and IDBA have the least number of erroneous bases. Only 10 contigs (4437 bases) constructed by IDBA-UD and 3 contigs (3301 bases) by IDBA cannot be aligned to the genome reference, compared to 1079 contigs (112505 bases) by SOAPdenovo and 5 contigs (15921 bases) by Velvet. Among aligned contigs, all assemblers have similar substitution error rate.

IDBA-UD also constructed the longest scaffolds with N50 being 194322, which is nearly twice of N50 of scaffolds constructed by other assemblers. Although N50 of scaffolds generated by different assemblers are similar, the coverage of scaffolds generated by all assemblers are much lower than that of contigs except IDBA-UD. IDBA-UD made the least misassembly during the scaffolding process, because of having longer and more accurate contigs.

In general, IDBA-UD achieved its best performance by iterating k from 20 to 100, while Velvet, SOAPdenovo and IDBA had best performance when k is set to a small value (21, 31 and 20-40, respectively). Because the Local-Assembly procedure can reconstruct missing k-mers, IDBA-UD can iterate k to a large value to construct very long contigs. The other assemblers are not able to reconstruct missing k-mers so that a reasonably small k is used to balance the gaps and branches problem. The running time and memory cost are more or less the same among all assemblers.

k	29	39	49	59	69	79	89	99
All branches	5509	2117	1019	618	375	288	195	149
IDBA-UD (expected)	5507	2115	1018	616	373	284	190	142
IDBA-UD	5328	2025	962	557	331	223	129	98
IDBA(expected)	5499	2103	1000	584	329	215	103	33
IDBA	5298	1986	933	513	280	156	65	32
SOAPdenovo	5259	1861	795	282	41	27	20	0
Velvet	3515	1356	617	219	35	27	20	0

Table. 3 Expected numbers of resolved branches by IDBA-UD, IDBA and the real number of all assemblers on the same data set in Table 2.

3.3 Local Assembly

The expected number of resolved branches (Theorem 3 in Appendix) by IDBA-UD, IDBA and the actual numbers by all assemblers for different repeat lengths k are shown in Table 3. We ran all assemblers with a specific k (k_{max}) and measured repeats with length [k – 10, k). Because SOAPdenovo and Velvet cannot handle even values of k due to the palindrome problem, 29, 39, etc are considered in Table 3. The number of resolved branches by IDBA-UD is slightly smaller than the expected number because some of the (k+2)-mers are removed as dead-end. As the k value increases, the number of resolved branches drops and the number of wrong contigs also increases because of the missing (k+2)-mers. Comparing to other assemblers, IDBA-UD can resolve more repeats by increasing k and gets longer contigs.

3.3. Single cell assembly

Two single cell short read sequencing data set *E.coli* (lane 1) and *S.aureus* (Chitsaz, et al., 2011)[3] were used to test the performance of IDBA-UD, SOAPdenovo, Velvet and Velvet-SC[4]. Genome sequence of *Escherichia coli str. K-12 substr. MG1655* and *Staphylococcus aureus subsp. aureus USA300_FPR3757* was downloaded from NCBI were used as reference for validation. The statistics of the assembly results of different assemblers are summarized in Table 4 and Table 5.

3.3.1 De novo assembly of *E.coli*

According to (Chitsaz, et al., 2011), the average sequencing depth of single cell sequencing data of *E.coli* is about 600x and the reads are sampled very unevenly. For contigs, SOAPdenovo and Velvet have similar N50 (6428 and 7679); Velvet-SC has the second best N50 (34454), as it considers the property of uneven depth to remove errors; IDBA-UD has the longest N50 (82007), as it considers uneven relative depth to remove errors and uses local assembly to reconstruct missing k-mers in low-depth regions. The contigs

[3] http://bix.ucsd.edu/projects/singlecell/

[4] Velvet-SC was run after EULER error correction as the authors suggested. The assembly result we presented is slightly different from that in Velvet-SC paper, because we calculated the N50 for aligned contigs rather than all contigs.

	k	contigs						Scaffolds						time	mem
		No.	N50	Max len	Cov(%)	Sub.err	err # (len)	No.	N50	Max len	Cov(%)	Sub.err	err # (len)		
IDBA-UD	20-100	210	36513	201860	99.56	0.0225%	10 (4437)	83	194322	406269	99.55%	0.0218%	5(3784)	63s	432M
SOAPdenovo	31	3346	1584	8691	98.36	0.0572%	1079(112k)	147	121214	246514	92.50%	0.0483%	1087(283k)	31s	852M
Velvet	21	473	13761	48489	98.09	0.0323%	5(15k)	111	111871	225438	96.81%	0.0291%	6(67k)	43s	526M
IDBA	20-40	672	8350	37391	98.52	0.0164%	3(3301)	60	119931	308798	97.55%	0.0161%	3(9420)	24s	414M

Table. 2 The assembly results on simulated 10x lenght-100 reads of *Lactobacillus plantarum* (~3.3Mb) with 1% error rate.

	k	Contigs						Scaffolds						time	mem
		No.	N50	Max len	Cov(%)	Sub.err	err # (len)	No.	N50	Max len	Cov(%)	Sub.err	err # (len)		
IDBA-UD	40-100	187	82007	224018	95.01	0.0017%	87(347k)	148	98306	224018	95.00	0.0016%	73(346k)	35m	3.2G
SOAPdenovo	75	6008	6428	50965	92.42	0.0421%	693(335k)	4419	25244	118128	86.49	0.0448%	588(653k)	30m	19G
Velvet	45	1736	7679	68395	92.69	0.0095%	160(266k)	1707	7795	68395	92.59	0.0095%	154(272k)	91m	11G
Velvet-SC	55	372	34454	157931	92.74	0.0019%	78(279k)							46m	8.3G

Table. 4 The assembly results on real single cell sequencing data of *Escherichia coli* (~4.64Mb). The read length is 100, insert distance is about 215, and the average depth is about 600x.

	k	Contigs						Scaffolds						time	mem
		No.	N50	Max len	Cov(%)	Sub.err	err # (len)	No.	N50	Max len	Cov(%)	Sub.err	err # (len)		
IDBA-UD	60-100	122	87502	175236	94.46	0.0027%	14(247k)	78	100895	308777	93.54	0.0019%	17(271k)	29m	3.8G
SOAPdenovo	95	1005	12214	92978	96.63	0.0067%	791(220k)	696	22632	149602	91.44	0.0072%	746(364k)	11m	12G
Velvet	55	520	15800	67677	94.14	0.0043%	79(99k)	516	16038	67677	94.13	0.0042%	77(100k)	135m	15G
Velvet-SC	55	322	29719	221773	93.12	0.0042%	43(312k)							224m	41G

Table. 5 The assembly result on real single cell sequencing data of *Staphylococcus aureus* (~2.87Mb). The read length is 100, insert distance is about 265, and the average depth is about 2300x.

constructed by IDBA-UD also have the highest coverage. IDBA-UD and Velvet-SC have the least number of substitution errors. All assemblers constructed some contigs cannot be aligned to the reference. Some of them are really misassembled contigs, but the alignment of reads against contigs and reference showed that some non-aligned contigs are from regions with structure variations.

After scaffolding, all assemblers produced longer scaffolds and lower coverage, but the difference between contigs and scaffolds is not much except SOAPdenovo. SOAPdenovo increased the N50 from 6428 to 25244, but the coverage dropped from 92.42% to 86.49%. That means the uneven depth of single cell assembly makes the scaffolding very difficult so that assemblers either cannot construct long scaffolds or make many mistakes in scaffolding procedure. Since IDBA-UD produced very long contigs, although scaffolding did not connect many contigs, the scaffolds generated by IDBA-UD have the longest N50 and highest coverage.

3.3.2 De novo assembly of *S.aureus*

The sequencing depth of single cell sequencing data of *S.aureus* is about 2300x, much higher than that of *E.coli*. SOAPdenovo and Velvet performed better for data set with higher sequencing depth. The contig N50 of SOAPdenovo and Velvet became 12214 and 15800, about twice of that of *E.coli*. IDBA-UD and Velvet-SC had similar performance as before, and handle reads with uneven depth quite well. Generally, higher sequencing depth does not affect the quality of assembly result much. The scaffolding also increases N50 and reduces coverage. The substitution error rates are very low for all assemblers for this high sequencing depth.

SOAPdenovo is the fastest assemblers among four assemblers. IDBA-UD took about twice of time of SOAPdenovo to perform de novo single cell assembly. The memory cost of IDBA-UD is much less than the others, as it did the filtering on *k*-mers. Depending on the nature of assemblers, different assemblers achieved its best performance with different *k* values. SOAPdenovo got its best performance by using a relatively large *k* (75 and 95) to reduce errors in high depth regions and introducing more gaps problem at the same time. It then relies paired-end information to connect contigs to form scaffolds. Therefore, it produced more and shorter contigs and more gaps in its scaffolds than the others. Velvet preferred a relatively small *k* (45 and 55) value, probably because it contains a very sophisticated algorithm to remove the errors in de Bruijn graph. Velvet-SC had best performance with a moderate *k* (55) value and relies on iteratively removing low depth erroneous contigs to form long contigs. IDBA-UD is able to iterate *k* values from small to large to build a de Bruijn graph with less gaps and less branches and obtain the best performance, through local assembly producing missing *k*-mers and iterate depth for reducing the errors.

3.4. Metagenomic assembly

3.4.1 De novo assembly of simulated metagenomics data

To evaluate the performance of IDBA-UD on metagenomic data, we considered a simulated data set with extremely uneven depth. This data set was synthesized by combining simulated reads of three species *Lactobacillus plantarum* (~3.3Mb), *Lactobacillus delbrueckii* (~1.85Mb) and *Lactobacillus reuteri F275 Kitasato* (~2Mb) from the same genus. Length-100 reads were sampled from these three species with sequencing depth 10x (low depth), 100x (moderate depth) and 1000x (high depth) respectively with 1% error rate. The simulated paired-end reads have an insert distance following normal distribution *N*(500, 50). IDBA-UD, SOAPdenovo, Velvet and Meta-IDBA were executed on this simulated metagenomic sequencing data set. Because the depth is highly uneven, the Pre-Error-Correction of IDBA-UD was activated to remove errors. The experiment results are showed in Table 6. In addition to the statistics we presented before, the sequencing depth of each species was considered for comparison.

	k	contigs							Scaffolds							time	mem
		No.	N50	Cov(%)	10x	100x	1000x	err # (len)	No.	N50	cov	10x	100x	1000x	err # (len)		
IDBA-UD	20-100	546	44879	99.15	99.31	98.48	99.52	9(4k)	335	104696	97.80	99.23	95.90	97.21	10(69k)	35m	6.9G
SOAPdenovo	45	16266	461	79.57	95.45	95.39	39.40	1347(166k)	10036	1356	74.01	83.43	95.37	39.28	1404(641k)	30m	18G
Velvet	45	12018	418	72.73	94.66	95.24	16.66	6200(710k)	11976	418	72.58	94.34	95.24	16.66	6219(724k)	91m	18G
Meta-IDBA	20-100	2636	4588	89.62	85.29	86.96	99.09	127(292k)								46m	6.2G

Table. 6 The assembly result on simulated metagenomic data set of *Lactobacillus plantarum* (~3.3Mb), *Lactobacillus delbrueckii* (~1.85Mb) and *Lactobacillus reuteri F275 Kitasato* (~2Mb). The sequencing depth of these species are 10x, 100x and 1000x respectively. The read length is 100, error rate is set to 1% and the insert distance follows normal distribution *N(500, 50)*.

	k	contigs				Scaffolds				Gene #	time	mem
		No.	N50	max len	Total Length	No.	N50	max len	Total Length			
IDBA-UD	20-100	39221	18658	39221	18658	37512	20737	673651	98243412	66298	138m	11G
SOAPdenovo	55	99213	2066	113678	92059177	93961	4575	319705	90788489	41093	61m	31G
Velvet	55	33165	3476	171410	65630428	30419	5065	358885	65887174	40914	176m	35G
Meta-IDBA	20-100	20-100	19978	7710	325768					48693	49m	12G

Table. 7 The assembly results on human gut microbial short read data (SRR041654 and SRR041655) from NCBI. The read length is 100 and the insert distance is 260.

	k	contigs							Scaffolds						
Genus		No.	N50	Cov(%)	10x	100x	1000x	err # (len)	No.	N50	cov	10x	50x	250x	err # (len)
IDBA-UD	20-100	713	177198	94.90	93.16	96.06	95.52	6(81k)	394	318240	87.72	81.41	89.66	92.13	14(593k)
SOAPdenovo	45	20427	6438	80.40	85.87	89.14	64.76	184(24k)	12938	155694	72.13	69.16	84.25	59.97	551(883k)
Velvet	45	21904	660	91.69	91.72	94.96	88.70	1334(179k)	21904	660	91.69	91.72	94.96	88.70	1334(179k)
Meta-IDBA	20-100	2283	26504	86.38	74.25	90.20	95.19	143(551k)							
Family		No.	N50	Cov(%)	10x	100x	1000x	err # (len)	No.	N50	cov	10x	50x	250x	err # (len)
IDBA-UD	20-100	904	163720	98.98	97.96	99.28	99.94	6(35k)	475	393540	96.45	96.58	98.28	95.06	8(119k)
SOAPdenovo	45	23908	6673	88.74	93.65	98.44	71.54	195(29k)	14449	160345	80.73	73.21	97.59	70.37	419(1160k)
Velvet	45	24785	795	95.48	93.03	97.82	95.83	1208(171k)	24785	795	95.48	93.03	97.82	95.83	1208(171k)
Meta-IDBA	20-100	3014	22311	80.53	66.38	88.13	89.96	233(1057k)							
Class		No.	N50	Cov(%)	10x	100x	1000x	err # (len)	No.	N50	cov	10x	50x	250x	err # (len)
IDBA-UD	20-100	488	236177	99.64	99.30	99.94	99.62	3(18k)	253	849606	96.42	95.30	98.76	97.59	3(60k)
SOAPdenovo	45	20111	7319	90.34	95.28	99.60	70.97	157(24k)	11610	329675	85.84	84.65	99.57	69.98	290(601k)
Velvet	45	20736	766	96.27	94.49	99.30	94.79	1025(148k)	20736	766	96.27	94.49	99.30	94.79	1025(148k)
Meta-IDBA	20-100	2657	43332	90.19	80.76	96.74	94.41	86(398k)							

Table. 8 The assembly results on simulated metagenomic data sets in different taxonomic levels. For each level, 5 test cases with three randomly selected species are generated for testing. The depths of three species are set to 10x, 50x and 250x respectively. The read length is 100, error rate is set to 1% and the insert distance follows normal distribution *N(500, 50)*. The values presented in this table are results average 5 test cases.

SOAPdenovo and Velvet did not perform well on this simulated metagenomic data set. N50 of SOAPdenovo and Velvet are only 461 and 418 respectively. The contigs constructed by SOAPdenovo and Velvet covered most regions of moderate-depth species (95.39% and 95.24%) and low-depth species (95.45% and 94.66%) but a small portion of high-depth regions (39.40% and 16.66%). It is because they make a tradeoff between low-depth and high-depth regions and cannot handle them together. SOAPdenovo and Velvet have the best performance in N50 and coverage when k = 45. Small k is chosen because the genome size of low-depth species (~3.3Mb) is larger than high-depth species (~2Mb). In fact, their performance for k = 55 are similar except that the coverage of low-depth species decreases, the coverage of moderate-depth species remains the same and the coverage of high-depth species increases. Meta-IDBA designed for metagenomic assembly iterates k from small to large to capture both low-depth and high-depth regions. At each iteration, missing k-mers in low-depth and moderate-depth regions introduce fragmentation in the assembly result. Thus, Meta-IDBA outperforms SOAPdenovo and Velvet, but performs worse than IDBA-UD. The contigs constructed by Meta-IDBA covered over 99% of the regions in high-depth species, slightly less than 90% of the regions in low-depth species and moderate-depth species.

As expected, IDBA-UD outperforms SOAPdenovo, Velvet and Meta-IDBA in all aspects. N50 (44879) of contigs constructed by IDBA-UD is about 10 times of the second best N50 (4588) of contigs constructed by Meta-IDBA. IDBA-UD also has the best coverage (99.15%), about 10% higher than the second highest by Meta-IDBA. The contigs constructed by IDBA-UD covered almost all the region of three species, the uneven depth does not affect the assembly quality of IDBA-UD. Similar to single cell assembly, scaffolding in all assemblers produced longer contigs but lower coverage. As for substitution error rate, IDBA-UD and Meta-IDBA have much higher accuracy and IDBA-UD constructed the least number of misassembled contigs. The running times of all assemblers are similar, but IDBA-UD and Meta-IDBA used about half of memory as SOAPdenovo and Velvet.

3.4.2 De novo assembly of human gut microbial short read data

Real human gut microbial sequencing data was used to assess the performance of IDBA-UD. The data set (SRR041654 and SRR041655) were downloaded from NCBI for assembly. The reads were generated by Illumina Genome Analyzer II with read length 100 and insert distance 260. IDBA-UD, SOAPdenovo, Velvet and Meta-IDBA were compared on this data set (The Pre-Error-Correction of IDBA-UD was activated.). Because there is no reference, we used the largest total contig size of all assemblers as the estimated genome size (98407199) for N50 calculation, and cannot analyze completeness of assembly by comparing genome coverage. MetaGeneAnnotator (Noguchi, et al., 2008)[5] was applied to the output of each assembler to predict the number of genes recovered. The statistics of assembly results are summarized in Table 7.

The contigs of SOAPdenovo and Velvet have similar N50, while SOAPdenovo produced much more contigs than Velvet. The total contig size of SOAPdenovo is 92059177. Meta-IDBA produced the smallest number of contigs, but N50 of contigs constructed by Meta-IDBA is larger than SOAPdenovo and Velvet. According to the analysis of the assembly result of simulated data, it is probably because Meta-IDBA reconstructed most of high-depth regions but missed some low-depth regions. IDBA-UD has the largest total contig size and the highest N50 (18658). The contigs constructed by IDBA-UD contain the largest number of predicted genes (66298), which is 50% more than that of SOAPdenovo and Velvet. The results reveal that IDBA-UD can assemble metagenomic data better than all the other assemblers. The running time of IDBA-UD is between SOAPdenovo and Velvet. The memory cost of IDBA-UD and Meta-IDBA is also about half of SOAPdenovo and Velvet.

3.4.3 De novo assembly of simulated metagenomic data with different similarity level

To show the performance of IDBA-UD on data with different similarity level, we constructed three kinds of data sets with genomes in same genus, family and class. For each similarity level, 5 test cases with three species are selected randomly and sampled with low depth (10x), middle depth (50x) and high depth (250x). The average of assembly results are shown in Table 8.

In general, all assemblers have the best performance on data set in class level and the worst performance on data set in genus level. Contigs and scaffolds generated by IDBA-UD have the largest N50 and highest coverage in all three kinds of data sets. Velvet generated the second highest coverage, but it produced the shortest N50. The scaffolds generated by SOAPdenovo have the second largest N50, but they covered the least portion of genomes. Meta-IDBA is somehow in the middle, because it is designed to handle similar subspecies problem rather than different expression levels. In the assembly results of IDBA-UD, all species got similar coverage. All the other assemblers generated high coverage for middle-depth (50x) species but lower coverage for low-depth and high-depth species, because they balanced the gap problems in low-depth species and high-depth species and benefited the middle-depth species. Consistent with previous experiments, IDBA-UD outperformed all the existing assemblers in sequencing data with highly uneven depth in all these experiments.

4 DISCUSSION AND CONCLUSION

In this paper, we proposed a new assembler IDBA-UD, an extension of IDBA, to assemble short sequencing reads with highly uneven depth. Besides iterating k from small to large, IDBA-UD reconstructs missing k-mers by local assembly and removes errors by iteratively removing low-depth contigs. The experiment results on both simulated and real data sets showed that IDBA-UD outperformed all existing assemblers in assembling data sets with highly uneven depth. For metagenomic data, there are more common k-mers between genomes from subspecies of the same species than genomes from different species. This information is used in Meta-IDBA for assembling metagenomic data. As a future work, we should study how to integrate this information in IDBA-UD for better performance.

ACKNOWLEDGEMENTS

This work was supported in part by HKU GRF funding (HKU 7116/08E, HKU 719709E, HKU 7111/12) and HKU Genomics SRT funding.

REFERENCES

Butler, J., *et al.* (2008) ALLPATHS: de novo assembly of whole-genome shotgun microreads, *Genome Res*, **18**, 810-820.

Chaisson, M.J., Brinza, D. and Pevzner, P.A. (2009) De novo fragment assembly with short mate-paired reads: Does the read length matter?, *Genome Res*, **19**, 336-346.

Chaisson, M.J. and Pevzner, P.A. (2008) Short read fragment assembly of bacterial genomes, *Genome Res*, **18**, 324-330.

Chitsaz, H., *et al.* (2011) Efficient de novo assembly of single-cell bacterial genomes from short-read data sets, *Nat Biotechnol*, **29**, 915-921.

Earl, D., *et al.* (2011) Assemblathon 1: A competitive assessment of de novo short read assembly methods, *Genome Res*, **21**, 2224-2241.

Hernandez, D., *et al.* (2008) De novo bacterial genome sequencing: millions of very short reads assembled on a desktop computer, *Genome Res*, **18**, 802-809.

Kelley, D.R., Schatz, M.C. and Salzberg, S.L. (2010) Quake: quality-aware detection and correction of sequencing errors, *Genome Biol*, **11**, R116.

Kent, W.J. (2002) BLAT--the BLAST-like alignment tool, *Genome Res*, **12**, 656-664.

Li, R., *et al.* (2010) De novo assembly of human genomes with massively parallel short read sequencing, *Genome Res*, **20**, 265-272.

Medvedev, P., *et al.* (2011) Error correction of high-throughput sequencing datasets with non-uniform coverage, *Bioinformatics*, **27**, i137-141.

Noguchi, H., Taniguchi, T. and Itoh, T. (2008) MetaGeneAnnotator: detecting species-specific patterns of ribosomal binding site for precise gene prediction in anonymous prokaryotic and phage genomes, *DNA Res*, **15**, 387-396.

Peng, Y., *et al.* (2011) Meta-IDBA: a de Novo assembler for metagenomic data, *Bioinformatics*, **27**, i94-101.

Peng, Y., *et al.* (2010) IDBA- A Practical Iterative de Bruijn Graph De Novo Assembler. *RECOMB*. Lisbon.

Pevzner, P.A., Tang, H. and Waterman, M.S. (2001) An Eulerian path approach to DNA fragment assembly, *Proc Natl Acad Sci U S A*, **98**, 9748-9753.

Rodrigue, S., *et al.* (2009) Whole genome amplification and de novo assembly of single bacterial cells, *PLoS One*, **4**, e6864.

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

Wooley, J.C., Godzik, A. and Friedberg, I. (2010) A primer on metagenomics, *PLoS Comput Biol*, **6**, e1000667.

Woyke, T., *et al.* (2009) Assembling the marine metagenome, one cell at a time, *PLoS One*, **4**, e5299.

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

[5] Only complete genes predicted by MetaGeneAnnotator are considered as recovered.

Zerbino, D.R., *et al.* (2009) Pebble and rock band: heuristic resolution of repeats and scaffolding in the velvet short-read de novo assembler, *PLoS One*, **4**, e8407.

Appendix

Branches may be caused by erroneous reads (*k*-mers), variations (SNPs) or repeats. The branches caused by erroneous *k*-mers can be solved by the graph structure, such as dead-end or bubbles. The branches caused by a length-*k* repeats can be resolved if we have a (*k*+2)-mers covering the repeat, which can be obtained if the corresponding (*k*+2)-mers covering the repeat are sampled in reads or the (*k*+2)-mer is obtained by local assembly.

Figure 3 shows an example of H_k and H_{k+1} for resolving a length-*k* repeat *v* and its associated branches. Because it is impossible to resolve these branches by H_k itself, reads and contigs are considered in each iteration from H_k to H_{k+1} for resolve them. If the (*k+2*)-mers covering repeat *v*, e.g., *uvw* and *u'vw'*, exist in reads, these (*k+2*)-mers can be used to convert branches to simple paths. If some of these (*k+2*)-mers are missing in reads due to low depth or errors, then local assembly can be used to reconstruct them as shown in Figure 2.

In this section, we try to calculate the expected number of branches caused by repeats that can be resolved by (*k*+2)-mers which are already existing in reads or reconstructed by locally assembly. Theorem 2 gives the expected number of (*k*+2)-mers covering a repeat being sampled *f* times (applied by IDBA). Theorem 3 gives the expected number of (*k*+2)-mers covering a repeat being sampled *f* times or reconstructed by local assembly (applied by IDBA-UD). Thus, the difference between these two expected numbers as given in Theorems 2 and 3 indicates the expected number of repeats resolved by local assembly.

Theorem 1: Assume *t* length-*l* reads are uniformly sampled from a length-*g* genome with error rate *e*, the probability that a *k*-mer *v* appearing *x* (g >> x) times in the genome being sampled at least *m* times is at least

$$P_{k,m,x} = 1 - \sum_{i=0}^{m-1} \binom{t}{i} p^i (1-p)^{t-i}$$

where $p = \dfrac{(l-k+1)x}{g-l+1} \times (1-e)^k$

Proof:

Pr(*v* is sampled in a read)

≤ Pr(a read containing *v* is sampled) Pr(*v* is sampled | a read containing *v* is sampled)

$= \dfrac{(l-k+1)x}{g-l+1} \times (1-e)^k$

The probability that a correct *k*-mer *v* appears less than *m* times is at most $\sum_{i=0}^{m-1} \binom{t}{i} p^i (1-p)^{t-i}$, so the result follows. □

Theorem 2: Assume *t* length-*l* reads are uniformly sampled from a length-*g* genome with error rate *e* and R_k is the set of repeats with length *k*. If the support requirement for resolving a branch is *f*, then the expected number of resolved branches S_k from *k* to *k*+1 is at least

$$\sum_{r \in R_k} \sum_{b = Y(r)} P_{k+2,f,x(b)}$$

where *Y*(*r*) is the set of the (*k*+2)-mers covering repeat *r* in the genome and *x*(*b*) is the number of times that *b* appear in the genome.

Proof:

In order to resolve a repeat of length *k*, a (*k*+2)-mer appearing at least *f* times in the reads is needed and the probability of such (*k*+2)-mer is $P_{k+2,f,x(b)}$ (Theorem 1). □

If reads are localized to a specific region of genome, then the branches caused by distant repeats may disappear and convert to simple paths. In this way, local assembly may generate local contigs which help to resolve branches.

Theorem 3: Assume *t* length-*l* reads are uniformly sampled from a length-*g* genome with error rate *e*, R_k is the set of repeats with length *k* and LR_k is the set of length-*k* repeats occurring at least twice in the genome within insert distance. If the support requirement for resolving a branch is *f* and IDBA-UD is applied on the data from *k* to *k*+1, then the expected number of resolved branches LS_k is at least

$$| R_k - LR_k | + \sum_{b \in LR_k} \sum_{b = Y(r)} P_{k+2,f,x(b)}$$

where *Y*(*r*) is the set of the (*k*+2)-mers covering repeat *r* in the genome and *x*(*b*) is the number of times that *b* appears in the genome.

Proof:

If a repeat does not appear twice within insert distance, then it can be resolved by IDBA-UD, i.e., $|R_k$ - $LR_k|$ and those repeats occurring more than once within the insert distance can only be resolved by the (*k*+2)-mers appearing at least *f* times in reads (Theorem 2).