Lawrence Berkeley National Laboratory

Title
BIGMAC : breaking inaccurate genomes and merging assembled contigs for long read metagenomic assembly.

Permalink
https://escholarship.org/uc/item/1jw9v58c

Journal
BMC Bioinformatics, 17(1)

Authors
Rao, Satish
Lam, Ka-Kit
Hall, Richard
et al.

Publication Date
2016-10-28

DOI
10.1186/s12859-016-1288-y

Peer reviewed

RESEARCH ARTICLE

CrossMark

BIGMAC : breaking inaccurate genomes and merging assembled contigs for long read metagenomic assembly

Ka-Kit Lam[1] , Richard Hall[2], Alicia Clum[3] and Satish Rao[1*]

Abstract

Background: The problem of de-novo assembly for metagenomes using only long reads is gaining attention. We study whether post-processing metagenomic assemblies with the original input long reads can result in quality improvement. Previous approaches have focused on pre-processing reads and optimizing assemblers. BIGMAC takes an alternative perspective to focus on the post-processing step.

Results: Using both the assembled contigs and original long reads as input, BIGMAC first breaks the contigs at potentially mis-assembled locations and subsequently scaffolds contigs. Our experiments on metagenomes assembled from long reads show that BIGMAC can improve assembly quality by reducing the number of mis-assemblies while maintaining or increasing N50 and N75. Moreover, BIGMAC shows the largest N75 to number of mis-assemblies ratio on all tested datasets when compared to other post-processing tools.

Conclusions: BIGMAC demonstrates the effectiveness of the post-processing approach in improving the quality of metagenomic assemblies.

Keywords: Genome assembly, Next generation sequencing, Metagenomics

Background

Introduction

De-novo assembly is a fundamental yet difficult [1] computational problem in metagenomics. Indeed, there is currently an open challenge for metagenomic assembly using short reads, titled "Critical Assessment of Metagenomic Interpretation (CAMI [2])." On the other hand, emerging sequencing technologies can provide extra information and make the computational problem more tractable. For example, long reads are increasingly being shown to be valuable in the de-novo assembly of single genomes [3]. Therefore, it is natural to study metagenomic assembly using long reads. Current assembly approaches for long reads focus on developing more optimized assemblers to leverage the power of the data. However, significant engineering effort is usually involved to build an end-to-end assembler.
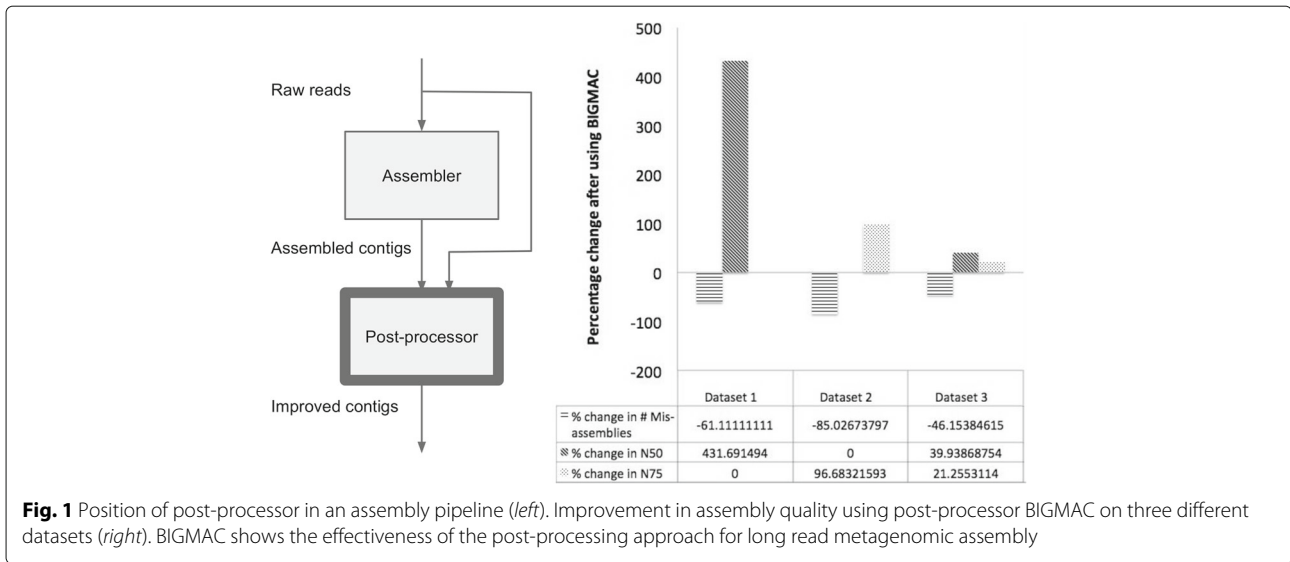
We take a different perspective, focusing the design effort on a post-processor that improves assembled contigs using the original long read data (Fig. 1). The main question is whether we can achieve quality improvement with this approach using typical long reads. This post-pocessing approach is attractive because it leverages existing software. Consequently, we can focus design effort and computational resources to specifically address thorny issues arising from the nature of new data, complex repeats, varying abundances and noise. Moreover, since the long reads are part of the input, the post-processor can bring back information missed upstream. In single genome assembly, FinisherSC [4] has demonstrated the effectiveness of this approach. In this paper, we investigate the effectiveness of this post-processing approach for metagenomic assembly.

BIGMAC is a post-processor to improve metagenomic assemblies with long read only data. It first breaks contigs at potentially mis-assembled locations and subsequently scaffolds contigs. In our experiments, BIGMAC

*Correspondence: satishr@cs.berkeley.edu
[1]Department of Electrical Engineering and Computer Sciences, UC Berkeley, Berkeley, USA
Full list of author information is available at the end of the article

Lam *et al. BMC Bioinformatics* (2016) 17:435

Page 2 of 11

Fig. 1 Position of post-processor in an assembly pipeline (*left*). Improvement in assembly quality using post-processor BIGMAC on three different datasets (*right*). BIGMAC shows the effectiveness of the post-processing approach for long read metagenomic assembly

demonstrates promising results on several mock communities using data from the Pacific Biosciences long read sequencer. Inputs to BIGMAC include assembled contigs from HGAP [5] and the original raw reads with adapters removed. After assembly and post-processing, we use QUAST [6] to evaluate and compare the assembly quality of contigs before and after using BIGMAC. As shown in Fig. 1, BIGMAC improves the quality of the contigs by reducing the number of mis-assemblies while maintaining/increasing N50 and N75. This demonstrates the effectiveness of the post-processing approach for metagenomic assembly with long reads.

A top-down design of BIGMAC
We use a hypothetical yet representative set of input data to illustrate the design of BIGMAC in a top-down manner. Let g_1, g_2 be two genomes of abundances λ_1, λ_2 respectively. Assume that they share a long repeat in the middle, that is, $g_1 = x_1 r y_1, g_2 = x_2 r y_2$. Unfortunately, an upstream assembler mis-assembles the reads and produces two contigs c_1, c_2 with incorrect joins at the repeat. That is, $c_1 = x_1 r y_2, c_2 = x_2 r y_1$. As an assembly post-processor, BIGMAC takes the mis-assembled contigs c_1, c_2 and original reads as input. Its goal is to reproduce g_1, g_2. To achieve this, we immediately recognize that there should be components for fixing mis-assemblies and scaffolding contigs. In BIGMAC, they are respectively Breaker and Merger. An illustration is given in Fig. 2.

Breaker is further divided into two subcomponents: Signal Detector and Signal Aggregator. Signal Detector collects signals that indicates mis-assemblies and Signal Aggregator subsequently makes decisions on breaking contigs based on the collected signals. In our example, the coverage fluctuation between λ_1, λ_2 along the contigs

c_1, c_2 and the presence of long repeat r are useful signals that Signal Detector collects. After aggregating these signals, Signal Aggregator decides on breaking both the contigs c_1 and c_2 at the starting points of the repeat r. Therefore, the output contigs of Breaker are $x_1, x_2, r y_1, r y_2$.
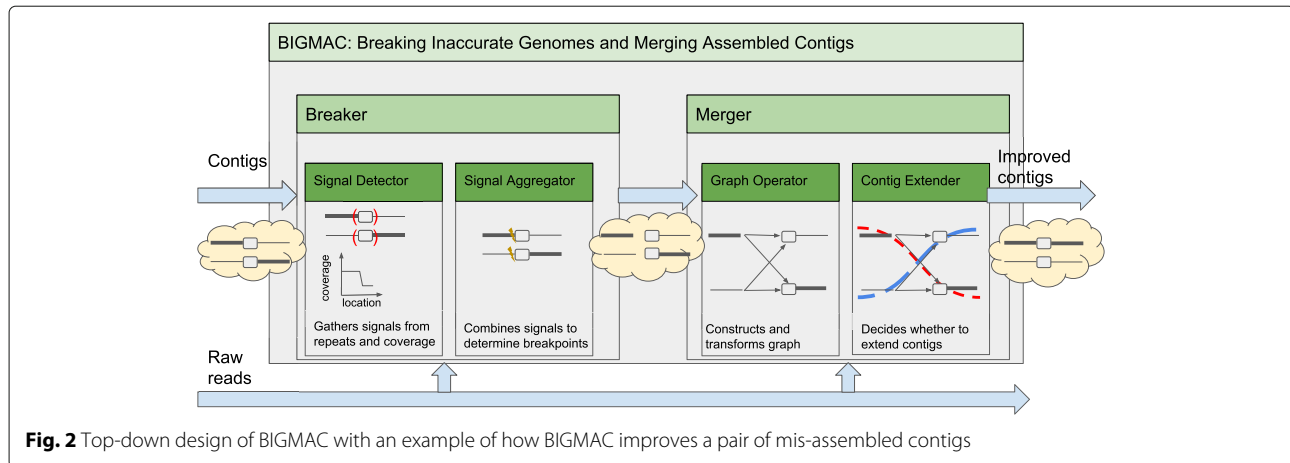
Merger is also divided into two subcomponents: Graph Operator and Contig Extender. With information from the original reads, Graph Operator establishes connectivity of the input contigs using string graphs. Then, based on the evidence from spanning reads and contig coverage, Contig Extender extends input contigs into longer contigs. In our example, the input contigs to Merger are $x_1, x_2, r y_1, r y_2$. Graph Operator forms a string graph with edges $x_1 \rightarrow r y_1, x_1 \rightarrow r y_2, x_2 \rightarrow r y_1$ and $x_2 \rightarrow r y_2$. Contig Extender analyzes the coverage depth of the related contigs and decides to merge contigs to form $x_1 r y_1$ and $x_2 r y_2$, thus reproducing the correct genomes.

Finally, we remark that BIGMAC uses overlap information between reads and contigs or among contigs. In our implementation, the overlap information is provided by running MUMmer [7] on appropriate strings. However, we note that one can implement the pipeline by replacing MUMmer with other aligners.

Methods
Breaker: breaking inaccurate genome
After the functional decomposition of BIGMAC in the previous section, we are ready to investigate its first component: Breaker. We note that the goal of Breaker is to fix mis-assemblies. In order to achieve that, we need to collect sensible signals related to mis-assemblies and subsequently aggregate the signals to make the contig breaking decisions.

Lam *et al. BMC Bioinformatics*   (2016) 17:435

Page 3 of 11

Fig. 2 Top-down design of BIGMAC with an example of how BIGMAC improves a pair of mis-assembled contigs

Signal Detector

Signal Detector collects three important signals related to mis-assemblies.

Palindrome We are interested in palindromes because of their relationship to a form of chimeric reads, the adaptor-skipped reads, which are common in today's long read technology [8]. Since assemblers get stuck at these chimeric reads, the palindrome pattern in reads propagates to the corresponding contigs. Thus, the pattern of palindrome is a strong signal indicating mis-assemblies, particularly when the palindrome is long. A string tuple (a, b) is called a wrapping pair if the reverse complement of a is a prefix of b or the reverse complement of b is a suffix of a. x is called a palindrome if it is the concatenation of a wrapping pair (a, b), that is $x = ab$. The wrapping length of x is $\max_{x=ab,(a,b) \text{ is a wrapping pair}} \min(a.length, b.length)$. For example, $x = ACGGCCG$ is a palindrome of wrapping length 3; $(a, b) = (ACGG, CCG)$ is a wrapping pair because the reverse complement of b is CGG, which is a suffix of a. Since the minimum length of a and b is $\min(4, 3) = 3$ and the wrapping length of x cannot exceed 3, the wrapping length for x is 3.

   Signal Detector collects information about palindromes by aligning each contig against itself. It then identifies palindromes with long wrapping length because that indicates mis-assemblies. The corresponding palindromes' information is then put into $S_{palindrome}$. To improve sensitivity, Signal Detector allows approximate match when searching for palindromes. We note that approximate matches are determined by computing the edit distance of the corresponding strings. To determine approximate matches in BIGMAC, we use nucmer in MUMmer [7] with default parameters and with option –maxmatch. Two strings are considered as approximately matched when nucmer reports so.

Repeat Since long repeats confuse assemblers, their endpoints are possible candidates for mis-assemblies. Let $s_1 = axb, s_2 = cxd$, a repeat between s_1, s_2 is specified by the endpoints of x in s_1, s_2. For example, $s_1 = CAAAAT, s_2 = GAAAAG$, the endpoints of the repeat $AAAA$ are the position specified by ! in $C!AAAA!T, G!AAAA!G$. Signal Detector aligns contigs against themselves to find the repeats. It then marks down the positions of the endpoints and puts them in a set called S_{repeat}. To improve sensitivity, Signal Detector allows approximate matches when searching for repeats. We note that approximate matches are determined by computing the edit distance of the corresponding strings. Moreover, it only considers repeats that are maximal and are of significant length.

Coverage Significant coverage variation along contigs can correspond to false joins of sequences from different genomes with different abundances. Coverage profile is the coverage depth along the contigs. For example, the coverage profile along a string $s = ACGT$ is $(1, 2, 2, 1)$ if the reads are AC, CG, GT. Signal detector aligns original reads to the contigs to find the coverage profile, which is called $S_{coverage}$.

Signal Aggregator

After Signal Detector collects signals regarding palindromes $S_{palindrome}$, repeats S_{repeat} and coverage profile $S_{coverage}$, Signal Aggregator uses them to determine the breakpoints on the input contigs C. The algorithm is summarized in Algorithm 1.

ChimericContigFixing The goal of ChimericContigFixing is to fix the contigs formed from chimeric reads. We simply break the palindromes in $S_{palindrome}$ at locations corresponding to their wrapping lengths. After removing redundant contigs, ChimericContigFixing returns the broken palindromes with the unchanged contigs.

Lam *et al. BMC Bioinformatics* (2016) 17:435

Page 4 of 11

Algorithm 1 Signal Aggregator

1: **Input:** Input contigs C and signals from Signal Detector $S_{palindrome}$, S_{repeat} and $S_{coverage}$

2: **Output:** Contigs C'' with less mis-assemblies

3: **procedure** (SignalAggregation)$S_{palindrome}$, S_{repeat}, $S_{coverage}$, C

4:     C' = ChimericContigFixing($S_{palindrome}$, C)
    ▷ Fix chimeric contigs

5:     S_{filter} = LocatePotentialMisassemblies(S_{repeat}, C')
    ▷ Locate mis-assemblies caused by repeats

6:     C'' = ConfirmBreakPoints(S_{filter}, $S_{coverage}$, C')
    ▷ Confirm mis-assemblies using coverage

7:     **return** C''

LocatePotentialMisassemblies The goal of the subroutine LocatePotentialMisassemblies is to locate potential mis-assemblies caused by repeats. We study the design of this subroutine in this section.

Motivating question and example We can declare all the endpoints of approximate repeats in S_{repeat} to be potential mis-assemblies. While this is a sensible baseline algorithm, it is not immediately clear whether it is sufficient or necessary. It is thus natural to consider the following question.

Given a set of contigs, how can we find the smallest set of locations on contigs to break so that the broken contigs are consistent with any reasonable ground truth? To illustrate our ideas, we consider an example with contigs $x_1 = abcde, x_2 = fbcg, x_3 = hcdi$ with $\{a, b, c, d, e, f, g, h, i\}$ being strings of equal length L.

The baseline algorithm of breaking contigs at the starting points of all the long($\geq 2L$) repeats breaks the contigs 4 times (i.e. $a|b|cde, f|bcg, h|cdi$). However, interestingly, we will show that one only need to break the contigs 3 times to preserve consistency (i.e. $x_1 = ab|cde, x_2 = fb|cg, x_3 = h|cdi$) and that is optimal.
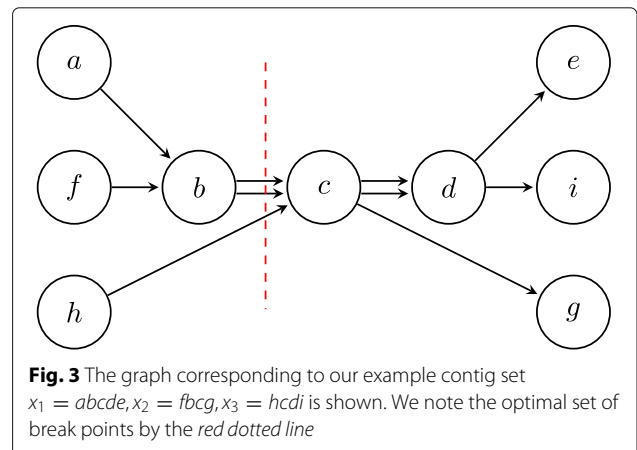
Modelling and problem formulation We will formalize the notions of feasible break points, feasible ground truth, consistency between sets of contigs, sufficiency of break points to achieve consistency and the optimiality criterion.

We use a graph theoretic framework. Specifically, we study a directed graph $G = (V, E)$ with m sources S and m sinks T where $\forall v \notin S \cup T$, $indeg(v) = outdeg(v)$ and parallel edges between two vertices are allowed. This is used to model a fully contracted De Bruijn graph formed by successive K-mers of the contigs. Vertices V are substrings of the contigs and edges E correspond to potentially mis-assembled locations on contigs. In our example, the set of vertices is $V = \{a, b, c, d, e, f, g, h, i\}$ and the set of edges is $E = \{ab, fb, bc_1, bc_2, hc, cd_1, cd_2, cg, de, di\}$. We

use subscripts to differentiate parallel edges joining the same vertices. The graph corresponding to our running example is shown in Fig. 3.

Given such a graph G, we note that E is the set of all **feasible break points** because each edge in the graph corresponds to a potentially mis-assembled location on contigs. A **feasible ground truth** corresponds to a set of m edge-disjoint source-to-sink trails that partitions the edge set E. For simplicity, we represent a trail as a sequence of the vertices in G, where the edges linking the vertices are ignored. For example, $\{abcde, fbcdi, hcg\}$ is a feasible ground truth while $\{abcg, fgde, hcdi\}$ is another feasible ground truth. The set of all feasible ground truths is GT.

We recall that our high level goal is to choose a set of feasible break points $R \subseteq E$ so that the broken contigs are consistent with any feasible ground truth. So, we need to define the notion of broken contigs and consistency between two sets of contigs under the current framework. Let $gt \in GT$, we denote $gt \backslash R$ be a set of trails after the removal of the edge set R. In particular, for the original contig set $C \in GT$, $C \backslash R$ is the set of broken contigs for the set of feasible break points R. For example, if $R = \{bc_1, bc_2, hc\}$ and $C = \{abcde, fbcdi, hcg\}$, $C \backslash R = \{ab, cde, fb, cdi, h, cg\}$. To capture consistency between two sets of contigs, we use the following definition. Given two sets of trails T_1, T_2, we say that T_1 is **consistent** with T_2 if $\forall x \in T_1, \exists y \in T_2$ s.t. $x \subseteq y$ and $\forall x' \in T_2, \exists y' \in T_1$ s.t. $x' \subseteq y'$. We call R a **sufficient breaking set** with respect to (C, GT) if $\forall gt \in GT, C \backslash R$ is consistent with $gt \backslash R$. In other words, R is a set of feasible break points that allows the broken contigs to be consistent with any feasible ground truth. Although this notion of sufficient breaking set is a natural model of the problem, it depends on the original contig set C, which is indeed not necessary. Specifically, we show that we have an equivalent definition of sufficient breaking set without referring back to the original contig set. Let us call R a sufficient breaking set with respect to G,

Fig. 3 The graph corresponding to our example contig set $x_1 = abcde, x_2 = fbcg, x_3 = hcdi$ is shown. We note the optimal set of break points by the *red dotted line*

Lam *et al. BMC Bioinformatics* (2016) 17:435

Page 5 of 11

or simply a sufficient breaking set, if $\forall gt_1, gt_2 \in GT, gt_1 \backslash R$ is consistent with $gt_2 \backslash R$.

Proposition 1 *R is a sufficient breaking set with respect to (C, GT) if and only if R a sufficient breaking set with respect to G.*

Proof The backward direction is immediate because $C \in GT$. We will show the forward direction as follows. Let $g_1, g_2 \in GT$ and we want to show that $g_1 \backslash R$ is consistent with $g_2 \backslash R$. Since R is a sufficient breaking set with respect to $(C, GT), g_1 \backslash R$ is consistent with $C \backslash R$. Therefore, $\forall x \in g_1 \backslash R \exists y \in C \backslash R$ s.t. $x \subseteq y$. But since $g_2 \backslash R$ is consistent with $C \backslash R$, we have $\exists z \in g_2 \backslash R$ s.t. $y \subseteq z$. By transitivity, we have $x \subseteq y \subseteq z \in g_2 \backslash R$. By symmetry, we can also show that $\forall x' \in g_2 \backslash R \exists y' \in g_1 \backslash R$ s.t. $x' \subseteq y'$. Thus, $g_1 \backslash R$ is consistent with $g_2 \backslash R$. □

Now, we state our **optimization criterion**. The goal here is to minimize the cardinality of the sufficient breaking set, formally as Eq. 1.

$$OPT = \min_{R \subseteq E} |R| \text{ s.t. } R \text{ is a sufficient breaking set with respect to} G \tag{1}$$

We will show that if we solve Eq. 1 for our running example, the answer is 3. This thus shows that the baseline algorithm of breaking contigs at all starting points (in our example, there are 4 of them) of all long repeats is not optimal.

Proposition 2 *For our running example, OPT = 3.*

Proof First, by inspecting the 6 feasible ground truths in GT, we note $R = \{bc_1, bc_2, hc\}$ is a sufficient breaking set with respect to G. Second, we note that the edge set need to disconnect sources and sinks, otherwise, we can find $g_1, g_2 \in GT$ such that $g_1 \backslash R, g_2 \backslash R$ are inconsistent. This means $|R|$ need to be $\geq$ mincut of the graph, which is 3. □

Algorithm development and performance guarantee
Next we describe an algorithm that finds a sufficient breaking set with respect to G. Let us denote a boolean variable b_e on each edge $e \in E$, with $\vec{b} = (b_e)_{e \in E}$. For $v \in V$, $InEdges(v), OutEdges(v)$ are the sets of incoming edges and outgoing edges at v respectively. $Prev(v), Succ(v)$ are the sets of predecessor vertices and successor vertices of v respectively. Our algorithm solves the following minimization problem (Eq. 2) as a proxy to (Eq. 1).

$$\min_{r \subseteq \vec{b}: r = True \Rightarrow \Phi(\vec{b}) = True} |r| \tag{2}$$

where,

$$\Phi(\vec{b}) = \wedge_{v:|Prev(v)| \geq 2 \text{ and } |Succ(v)| \geq 2}$$
$$(\wedge_{e \in InEdges(v)} b_e \vee \wedge_{e \in OutEdges(v)} b_e) \tag{3}$$

In other words, it includes either all the incoming edges or all the outgoing edges for every vertices with at least 2 successors and at least 2 predecessors to R. We then seek R with minimum cardinality among the choices available. If G can be decomposed into connected components, we can optimize $\Phi(\vec{b})$ independently on each connected component. In our implementation, if the size of the connected component is not too large, we optimize the objective function by exhaustion. Beyond a certain threshold, we simply output a feasible solution without optimizing. Indeed, in our experiments on real data, most of the connected components are not that large and this step typically takes a few minutes. But we remark that for more complex applications, one can further optimize the algorithm. For example, one can first topologically sort the vertices and then use dynamic programming to solve Eq. 2 along the sorted vertices.

We note that the algorithm described gives an optimal solution for our running example. Moreover, we show performance guarantee of the algorithm as follows.

Proposition 3 *Solving* Eq. 2 *gives a sufficient breaking set R if the graph G is fully contracted.*

Proof Let R be the set of edges selected by the algorithm. For any two $gt_1, gt_2 \in GT$, we want to show that $gt_1 \backslash R$ and $gt_2 \backslash R$ are consistent. By symmetry, it suffices to prove that if $x \in gt_1 \backslash R$, then $\exists y \in gt_2 \backslash R$ s.t. $x \subseteq y$. If $|x| = 2$, it is immediate because every edge other than those in R are covered. If $|x| \geq 3$, we will show that it is also true using proof by contradiction. If $\forall y \in gt_2 \backslash R, x \not\subseteq y$, we can find a sub-trail $x' = (a_1, a_2, \ldots, a_k, a_{k+1})$ of x such that $\exists y' \in gt_2 \backslash R$ s.t. $x'' = (a_1, \ldots, a_k) \subseteq y'$ but $\forall y \in gt_2 \backslash R, x' \not\subseteq y$. This implies $\exists a^* \neq a_{k+1}$ s.t. $(x'', a^*) \subseteq z$ for some $z \in gt_2 \backslash R$. Since the edge $(a_k, a_{k+1}) \subseteq x \in gt_1 \backslash R$, we know that (a_k, a_{k+1}) is not in R. Similarly, $(a_k, a^*) \notin R$ because $(a_k, a^*) \subseteq y' \in gt_2 \backslash R$. But since $|Succ(a_k)| \geq 2$, the fact that our algorithm does not include $(a_k, a^*), (a_k, a_{k+1})$ in R implies that $|Pred(a_k)| = 1$, namely $Pred(a_k) = \{a_{k-1}\}$. Note that we are considering a fully contracted graph. So, the fact that a_{k-1} exists implies that $|Succ(a_{k-1})| \geq 2$. But our algorithm has not removed edge (a_{k-1}, a_k). This gives $|Pred(a_{k-1})| = 1$. Inductively, we get $|Pred(a_i)| = 1 \forall 2 \leq i \leq k$. But we know that $(a_k, a_{k+1}) \subseteq w$ for some $w \in gt_2 \backslash R$. Since the edges along $(a_1, \ldots, a_{k+1})$ are not in R, this gives, $x' = (a_1, \ldots, a_{k+1}) \subseteq w \in gt_2 \backslash R$. This contradicts the assumption about x'. □

Lam *et al. BMC Bioinformatics*   (2016) 17:435

Page 6 of 11

Proposition 4 *If the graph G is fully contracted DAG without parallel edges, then solving* Eq. 2 *returns a sufficient breaking set that is of optimal cardinality, OPT.*

Proof It suffices to show that for any sufficient breaking set R, $\forall v \in V$ where $|Succ(v)| \geq 2, |Pred(v)| \geq 2$, we have either $InEdges(v) \subseteq R$ or $OutEdges(v) \subseteq R$. We prove it by contradiction. If not, $\exists v \in V$ where $|Succ(v)| \geq 2, |Pred(v)| \geq 2$ but $InEdges(v) \nsubseteq R$ and $OutEdges(v) \nsubseteq R$. Because it is a DAG, it means we can find $gt_1 \in GT$ such that $\exists x, y, x', y'$ such that $(x, v, y) \in gt_1$ and $(x', v, y') \in gt_1$. Now consider gt_2 to be a clone of gt_1 except that it has $(x, v, y'), (x', v, y)$ instead of $(x, v, y'), (x', v, y)$. We note that $gt_2 \in GT$. And because there are no parallel edges, (x, v, y) is not a subset of any of the trails in gt_2. So, we find two distinct $gt_1, gt_2 \in GT$ such that gt_1, gt_2 are not consistent. This contradicts the fact that R is a sufficient breaking set. □

It is noteworthy that if we are given any set of contigs from any $gt \in GT$, we still obtain the same set of broken contigs after the operation of removal of redundant trails, RemoveRedundant (i.e. we eliminate the contigs in a set A to form a minimal subset $B \subseteq A$ in which $\forall x \neq y \in B, x \nsubseteq y$). This can be formalized as follows.

Proposition 5 *If R is a sufficient breaking set, then for any* $gt_1, gt_2 \in GT$, *RemoveRedundant$(gt_1 \backslash R) =$ RemoveRedundant$(gt_2 \backslash R)$.*

Proof Let $s_i = RemoveRedundant(gt_i \backslash R)$ for $i \in \{1, 2\}$. By symmetry, it suffices to prove that $s_1 \subseteq s_2$ $\forall x \in s_1 \subseteq gt_1 \backslash R, \exists y \in gt_2 \backslash R$, such that $x \subseteq y$. Note that we can also find some $x^* \in s_2$ such that $y \subseteq x^*$. This gives $x \subseteq y \subseteq x^*$. However, since we have no redundant trails in s_1, we get $x = x^*$. Thus $x = x^* \in s_2$. □

To apply BIGMAC to real data, we have to implement the described algorithm with some further engineering. This includes methods to tolerate noise, to handle double stranded nature of DNA, and to construct De Bruijn graph from the repeats. Interestd readers can refer to the Additional file 1 : Appendix for these implementation details.

ConfirmBreakPoints The goal of ConfirmBreakPoints is to utilize the coverage profile $S_{coverage}$ to confirm breaking decisions at potentially mis-assembled locations specified in S_{filter}. For the sake of simplicity, we now consider a string s of length L, and a set of positions $Y = \{y_i\}_{1 \leq i \leq k}$ of s which are potential mis-assemblies. Furthermore, let us assume that all mis-assemblies are caused by mixing genomes of different abundances. Using Y, we can partition s into segments $\{s_i\}_{0 \leq i \leq k}$ of lengths respectively as $\{\ell_i\}_{0 \leq i \leq k}$. We let x_i be the number of reads that start in segment s_i, which can be estimated from $S_{coverage}$. The question is how to classify the points in Y as true mis-assemblies or as fake mis-assemblies.

One can use heuristics, like comparing coverage depth difference before and after each y_i. However, this is not ideal. For example, if we have coverage depth before and after y_1 differing by 1X, we would expect it to be a much stronger signal for true mis-assembly if the lengths ℓ_0, ℓ_1 are of order of 100 K rather than of 100 and this cannot be seen by considering coverage depth difference alone. For the case of just two segments of equal length and if we assume the x_i's are independent Poisson random variables, there is a popular statistical test, C-Test [9], that can make the decision. Formally, if $x_1 \sim Poisson(m_1)$ and $x_2 \sim Poisson(m_2)$, then C-Test is a test to decide between the hypothesis of $H_0 : m_1 = m_2$ against $H_1 : m_1 \neq m_2$. C-Test first considers $x_1 + x_2$ to compute the decision boundary and it then decides whether to reject H_0 based on x_1/x_2 and the previously derived decision boundary. The intuition is that $x_1 + x_2$ corresponds to the amount of data, which determines the confidence of a statistical test. Thus, if $x_1 + x_2$ is large, a small perturbation of x_1/x_2 from 1 can still be very significant and can correspond to a confident rejection of H_0.

However, we still need to think carefully about how to apply C-Test to our problem. One method is to directly apply k independent C-Test on each of the junctions y_i. However, that method cannot take advantage of the information from neighboring segments to boost the statistical power at a junction. Therefore, we develop the algorithm IterativeCTest. IterativeCTest performs traditional C-Test but in multiple iterations. Initially, it directly applies k independent C-Test on each of the junctions y_i. Some of the segments are merged and we aggregate the counts from the merged segments to continue to the next C-Test at the remaining unmerged junctions in Y. This process is repeated until the algorithm converges. Finally, we use the breaking decisions from IterativeCTest to break the incoming contigs and return the fixed contigs.

Merger: merging assembled contigs
After fixing mis-assemblies, we are ready to study another pillar of BIGMAC: Merger. Merger establishes connectivity among contigs and subsequently makes decisions to extend contigs.

Graph operator
The goal of Graph Operator is to identify candidates for contig extension. It forms and transforms a string graph using the overlap information among original reads and contigs. It subsequently analyzes the graph to find candidates for contig extension. The overall algorithm is summarized in Algorithm 2.

Lam *et al. BMC Bioinformatics* (2016) 17:435

Page 7 of 11

Algorithm 2 Graph operator
1: **Input:** Contigs C and original reads R
2: **Output:** String graph G with information about candidates for contig extension
3: **procedure** GRAPHOPERATOR(R, C)
4:     M = Mapping(R, S)
            ▷ Obtain mapping among contigs and reads
5:     G = FormGraph(M)
            ▷ Form string graph to represent connectivity
6:     G.GraphSurgery(M)   ▷ Simplify graph
7:     G.FindExtensionCandidates()
            ▷ Identify candidates for contig extension
8:     **return** G

Mapping We obtain mapping among contigs and reads. This provides the fundamental building block to construct the connectivity relationship among contigs and reads.

FormGraph The goal of FormGraph is to establish connectivity among contigs. We use contig-read string graph as our primary data structure. Contig-read string graph is a string graph [10] with both the contigs and the reads associated with their endpoints as nodes. The size of the graph is thus manageable because most reads are not included in the graph. Then, we construct an overlay graph (which we call the contig graph) such that the nodes are the contigs with weights on nodes being the coverage depth of contigs. In the contig graph, there is an edge between two nodes if there is a sequence of reads between the associated contigs. With these data structures, we can switch between macroscopic and microscopic representation of the contig connectivity easily.

GraphSurgery GraphSurgery simplifies the contig graph. This includes removal of transitive edge and edge contraction.

For nodes u, v, w, if we have edges $u \rightarrow v, u \rightarrow w$ and $w \rightarrow v$, then we call $u \rightarrow v$ a transitive edge. We perform transitive reduction [10] on the contig graph to remove transitive edges. Removing these transitive edges prevents us from finding false repeats in the next stage. To improve robustness, there is a pre-processing step before transitive reduction. If the contig w is too short and there are no reads that form head-to-tail overlap between w, u or w, v, then we can have a missing edge for transitive reduction to operate properly. Thus, we add the missing edge (either from u to w or w to v) back when we find contigs and related reads that suggest the missing edge might be there.

An edge $u \rightarrow v$ is contractable when the outgoing degree of u and the incoming degree of v are both 1. We contract edges to fill gaps. Our experience with FinisherSC is that data are dropped in the assembler and so

reconsidering them as a post-processing step can potentially fill some gaps. However, there is a caveat. In establishing connectivity in contig-read string graph, we only use reads close to each contig's endpoints (as a way to save computation resources), we may miss some legitimate connections. Therefore, very long repeats prevent the detection of linkage of contigs, thereby allow contigs to be erroneously merged. To address that, if there exists contig w that is connected (by some reads) to a region close to the right end of u/left end of v, then we avoid contraction of $u \rightarrow v$.

FindExtensionCandidates FindExtensionCandidates identifies candidates for contig extension by identifying local patterns in the contig graph. One form of extension candidates is a pair of contigs that are connected without competing partners. This corresponds to the contigs joined by a contractable edge. Another form of extension is a set of contigs that are connected with competing partners. This corresponds to the contigs linked together in the presence of repeats. In the contig graph, the repeat interior can either be represented as a separate node or not. If the repeat interior is represented as a separate node, the local subgraph is a star graph with the repeat node at the center. Otherwise, the local subgraph is a bipartite graph consisting of competing contigs. After identifying the contigs associated with a specific repeat, we can then merge contigs in the next stage.

Contig extender

After the operations by Graph Operator, we have extracted the potential contig extension candidates from the contig graph. It remains to decide whether and how to merge the corresponding contigs. In a high level, Contig Extender aims at solving the Contig Merging Decision Problem.

Contig merging decision problem *Given a set of incoming contigs I and a set of outgoing contigs O whose coverage depth and read connectivity information is known. Decide how to form an appropriate bipartite matching between I and O.*

While we do not intend to rigorously define the statement of Contig Merging Decision Problem now, it is clear that appropriate matching corresponds to one that achieves high accuracy. Contig Extender carefully analyzes the read connectivity and contig coverage to make decisions on merging contigs. In the coming discussion, we first study an effective heuristic that captures the essence of the problem. After that, we will study how to rigorously define the Contig Merging Decision Problem in a mathematical form and suggest an EM-algorithm in solving that.

Heuristic in solving the contig merging decision problem When the cardinality of the set of incoming contigs I and the set of outgoing contigs O are both 1, a natural decision is to merge them. Thus, the focus here is to study the case when $|I| > 1$ or $|O| > 1$. We select the reads that uniquely span one contig a in the incoming set and one contig b in the outgoing set. If there are multiple such reads, then we decide that a, b should be joined together provided that there does not exist any paths of reads that lead a to other contigs in the outgoing set and similarly for b. Moreover, we perform similar tests using contig coverage. If the coverage depth between two contigs is very close, they will be declared to be a potential merge pair. Then, we test whether there are any close runner-ups. If not, they should be merged. To combine the decisions made using spanning reads and coverage depth, we have a subroutine that discards all conflicting merges. We find that this heuristic for decision making works quite well in our datasets. However, in the coming discussion, we will study how to make the extension decisions in a more principled and unified manner.

General framework in solving the contig merging decision problem The challenge for the Contig Merging Decision Problem is the tradeoff for many physical quantities (e.g. abundance, edit distance of reads, noise level, number of spanning reads, etc). We address this by defining a confidence score based on parameter estimation. For simplicity of discussion, we assume that k is the cardinality of both the set of incoming contigs and the set of outgoing contigs. The goal is to find the best perfect matching with respect to a confidence score.

Let M be a perfect matching of contigs among incoming and outgoing groups I and O. Under M, there are k merged contigs. Let the observation be the set of related reads $X = \{R_i \mid 1 \le i \le n\}$. We define the parameters $\theta = \{(\lambda_j, I_j)_{1 \le j \le k}\}$, where λ_j is the normalized abundance of contig j and I_j is genomic content of the contig j. Note that $\sum_{1 \le j \le k} \lambda_j = 1$. So, the parameter estimation problem can be cast as $s_M = \max_\theta P_\theta(X \mid M)$, where s_M is the confidence score of the matching M. Finally, the best perfect matching can be found by comparing the corresponding confidence score.

Note that we have hidden variables $Z = (Z_i)_{1 \le i \le n}$ which indicate the contigs that reads X are extracted from (i.e. $Z_i \in \{1, 2, \ldots, k\}$). If we assume the length of the contig j to be ℓ_j and q to be the indel noise level (i.e. probability of $1 - 2q$ to be remained unaltered at each location), then we can use an EM-algorithm to obtain an estimate of θ. Specifically, the expected value of the log likelihood function $E_{q(Z|X,\theta^{(t)})} \left[\log P_{\theta^{(t)}}(X, Z, \theta^{(t+1)}) \right]$ is

$$\sum_{1 \le i \le n} \sum_{1 \le j \le k} M^j(R_i, I^{(t)}) \left[\log \frac{\lambda_j^{(t+1)}}{\ell_j} + |R_i| \log(1 - 2q) \right.$$
$$\left. + d(R_i, I^{(t+1)}) \log \frac{q}{1 - 2q} \right] \tag{4}$$

where $M^j(R, I^{(t)}) = \delta_{j = argmin_j d(R, I_j^{(t)})}$ is the assignment of reads to the most similar contig (with tie breaking using $\lambda^{(t)}$), $d(A, B)$ is the best local alignment score, $I^{(t)} = (I_j^{(t)})_{1 \le j \le k}$ are the genomic contents of the contigs at iteration t and $\lambda^{(t)} = (\lambda_j)_{1 \le j \le k}$ are the estimated abundances at iteration t. By maximizing the log likelihood function with respect to $\theta^{(t+1)}$, we have the update formulas as

$$\lambda_{j^*}^{(t+1)} = \frac{\sum_{1 \le i \le n} M^{j^*}(R_i, I^{(t)})}{\sum_{1 \le j \le k} \sum_{1 \le i \le n} M^j(R_i, I^{(t)})} \tag{5}$$

$$I_{j^*}^{(t+1)} = \underset{I}{\operatorname{argmin}} \sum_{1 \le i \le n} M^{j^*}(R_i, I^{(t)}) d(R_i, I) \tag{6}$$

Note that the update of I_{j^*} requires multiple sequence alignment. In general, the problem is NP-hard [11]. However, for noisy reads extracted from several contigs, the problem is not as difficult. For example, in the case of pure substitution noise, an efficient optimal solution is a column-wise majority vote. Despite the elegance and feasibility of this approach, it is still computationally more intense than the heuristic. Therefore, in our implementation of BIGMAC, the heuristic is the default method used in Contig Extender. But we also provide an experimental version of the EM-algorithm which can be used when users specify `-option emalgo=True` in their commands.

Results
End-to-end experiments
Synthetic data
We verify that BIGMAC correctly improves the incoming contigs using the following synthetic data. We generate two synthetic species of different abundances ($\frac{2}{7}, \frac{5}{7}$). They are composed of random nucleotide sequences of length 5 M each and share a common segment of length 12 K. We uniformly sample indel noise corrupted reads of length 6 K from the genomes, with coverage depth of 20X and 50X respectively. We also artificially construct mis-assembled contigs by switching the genome segments at the 12 K repeat.

The contigs and the reads are passed through BIGMAC. BIGMAC breaks the contigs at the mis-assembled point and joins them back correctly. One can see the sample run on [12]. This is also the example that we discuss in the top-down design of BIGMAC.

Lam *et al. BMC Bioinformatics*   (2016) 17:435

Page 9 of 11

Real data

We test the performance of BIGMAC in improving metagenomic assembly on PacBio only data. We use different datasets of different characteristics. Dataset 1 consists of a mock community of 5 species [13], with genomes of high similarity. Dataset 2 consists of a mock community of 23 species [14], with genomes of diverse abundances. We also remark that we have tested BIGMAC on a third PacBio only dataset (Dataset 3): a real experiment involving Desulfuromonas biwabikus, D. soudanensis and some other unknown genomes. We note that we know the complete ground truth for the metagenomes in both Dataset 1 and 2 but only know part of the ground truth for Dataset 3. We take the output of HGAP and use the raw reads to improve them using BIGMAC. We observe that in these datasets, BIGMAC reduces the number of mis-assemblies while maintaining/increasing N50 and N75. The results of BIGMAC is summarized in Table 1, where the quantity of mis-assemblies is obtained from the QUAST reports. By default, QUAST's statistics are based on contigs of size >= 500 bp. Interested readers can refer to the Additional file 1: Appendix for more details of the assessment. The data is provided in our online distribution and users can reproduce the results by issuing a single command `python reproduce.py`.

Comparison with other post-processing tools
Synthetic data

We use the synthetic data in "Synthetic data" section to evaluate and benchmark BIGMAC, FinisherSC [4], SSPACE_LongRead [15] and PBJelly [16]. BIGMAC is the only tool among the tested tools that fix the mis-assembled contigs and merge them back correctly. Other tested tools output the same mis-assembled contigs.

Real data

We perform end-to-end testing to compare performance of different tools. The comparison is shown in Table 2. BIGMAC shows the largest N75/# Mis-assemblies on all three datasets and largest N50/# Mis-assemblies on two

Table 1 Performance evaluation of BIGMAC on several mock communities

Dataset	Method	NContig	# Mis-assembly	N50	N75
1	HGAP	130	18	818655	274801
1	BIGMAC	129	7	4352719	274801
2	HGAP	477	187	397611	38471
2	BIGMAC	344	28	397611	75666
3	HGAP	185	26	257044	82370
3	BIGMAC	140	14	359704	99878

BIGMAC consistently improves assembly quality by simultaneously increasing contig contiguity and decreasing the number of mis-assemblies

Table 2 Comparison of performance of BIGMAC with other post-processing tools for long read assemblies

Data	Method	# Mis	N50	N75	N50/# Mis	N75/# Mis
1	HGAP	18	818655	274801	45481	15267
	BIGMAC	7	4352719	274801	**621817**	**39257**
	FinisherSC	32	2531294	415024	79103	12970
	PBJelly	19	4642330	418480	244333	22025
	SSPACE_LR	32	4657611	493683	145550	15428
2	HGAP	187	397611	38471	2126	206
	BIGMAC	28	397611	75666	**14200**	**2702**
	FinisherSC	192	654163	43018	3407	224
	PBJelly	271	1585584	61775	5851	228
	SSPACE_LR	255	1568442	95133	6151	373
3	HGAP	26	257044	82370	9886	3168
	BIGMAC	14	359704	99878	25693	**7134**
	FinisherSC	25	996532	97964	39861	3919
	PBJelly	27	1103847	128718	**40883**	4767
	SSPACE_LR	43	1266912	290104	29463	6747

BIGMAC shows the largest N75/# Mis on all three datasets and largest N50/# Mis on two out of three datasets. We also remark that BIGMAC is the only tool that can improve N50 and N75 while reducing the number of mis-assemblies. Note that # Mis is an abbreviation for the number of mis-assemblies

out of three datasets. Indeed, in the only dataset that BIGMAC does not have the largest N50/#Mis-assemblies, the number of contigs(i.e. L50) beyond N50 is 7. Due to the small number of contigs, this particular measurement on that dataset may not be significant. We also remark that BIGMAC is the only tool that improves contiguity (N50 and N75) and the number of mis-assemblies.

Simulation and testing on independent components

We perform micro-benchmarking on individual components of BIGMAC. The settings and results are summarized as follows.

Analysis of LocatePotentialMisassemblies

We test our break point finding algorithm used in LocatePotentialMisassemblies of Breaker on the synthetic data of $x_1 = abcde, x_2 = fbcg, x_3 = hcdi$ discussed in the previous section. The algorithm succeeds in identifying the minimum number of break points as 3. Also, it succeeds in identifying the minimum number of break points as 2 in the presence of double stranded DNA, in the test case of $x_1 = abcd, x_2 = ec'b'f$, where b', c' are the reverse complement of b, c respectively.

Analysis of ConfirmBreakPoints

We test IterativeCTest used in ConfirmBreakPoints of Breaker on synthetic data. We simulate mis-assemblies and variation on abundances by generating a sequence of Poisson random variables and compare the performance

Lam *et al. BMC Bioinformatics* (2016) 17:435

Page 10 of 11

of the algorithms on the simulated data as follows. We generate a sequence of 100 numbers by 100 independent Poisson random variables. The Poisson random variables have parameters of either 20 or 50. To select the parameters, we simulate 100 steps of a two-states Markov chain with transition probability of 0.1. We then evaluate the performance of C-Test and IterativeCTest on finding the true transition points, which correspond to the junctions of mis-assemblies. Taking average from 100 rounds of simulation, the recall of both C-Test and IterativeCTest are 0.93, meaning that they both can identify almost all transition points. C-Test has precision of 0.75 while the precision of IterativeCTest is of 0.87, meaning that IterativeCTest can avoid more fake mis-assemblies.

Analysis of merger

We compare Merger with other tools on synthetic data as follows. We use a synthetic contig set $\{x_1, x_2, r, y_1, y_2\}$ where the ground truth genomes are $(x_1, r, y_1), (x_2, r, y_2)$. The coverage depth of (x_1, y_1) and (x_2, y_2) are 20X and 50X respectively. We pass the reads together with the contig set to FinisherSC, PBJelly, SSPACE_LongRead to perform scaffolding. We note that BIGMAC is the only tool the can scaffold the contigs correctly into 2 contigs by using the abundance information among the tested tools. Other tools either do not change the input or simply merge r with some of $\{x_1, x_2, y_1, y_2\}$, resulting in 4 contigs.

Runtime of BIGMAC

The runtime of BIGMAC is summarized in Table 3. We use 20 threads to run the tool on a server computer. The server computer is equipped with 64 AMD Opteron(tm) Processor 6380(8 cores) with frequency 2500 MHz and 362 GB RAM. We note that the majority of the time is spent on alignment of contigs and reads by MUMmer.

Discussion

There are two main implications from the experiments performed. First, we show that post-processing assemblies

Table 3 Runtime of BIGMAC and the corresponding file size

Dataset	Component	Contig file size (MB)	Read file size (GB)	Running time (sec)
Synthetic	Breaker	9.6	0.335	164
Synthetic	Merger	9.6	0.335	123
1	Breaker	30	5.7	6646
1	Merger	29	5.7	6998
2	Breaker	32	5.8	4865
2	Merger	29	5.8	5087
3	Breaker	17	7.6	7099
3	Merger	14	7.6	6887

is a feasible alternative in improving assembly quality to building another assembler from scratch. This is demonstrated by BIGMAC showing simultanous improvement in terms of number of mis-assembly and contiguity. We note that this is a non-trivial feature because all other tested tools fail to achieve it. Second, BIGMAC is competitive when compared to the existing post-processing tools. In particular, it shows better N75/# Mis-assemblies than all other tested tools in all tested datasets. Moreover, BIGMAC is also the only tool that can handle abundance information, which makes it an attractive candidate for improving metagenomic assembly.

Future work

We remark that the creation of BIGMAC sheds light on many interesting future direction. For example, it would be interesting to apply similar ideas to hybrid data, which has a lot of potential in the context of metagenomics. Moreover, it would also be useful to try BIGMAC and other post-processing methodology on more real data to better characterize the approach.

Conclusion

We study an agile approach in developing de novo metagenomic assemblers: post-processing metagenomic assemblies with original input data. BIGMAC demonstrates the effectiveness of the post-processing approach in improving the quality of metagenomic assemblies using long reads. BIGMAC thus serves as an example that developing post-processors is a good alternative to building end-to-end assemblers, which typically involves more engineering efforts.

Additional file

Additional file 1: Supplementary materials include implementation details, data analysis, theoretical analysis and commands used to run different tools. (PDF 492 kb)

Availability of data and materials
Information of the software BIGMAC is as follows.

- Project name: BIGMAC
- Project home page: https://github.com/kakitone/BIGMAC
- Archived version: Not applicable
- Operating system(s): Linux or Mac OS
- Programming language: Python
- Other requirements: MUMmer 3.23,
- Any restrictions to use by non-academics: Not applicable

Lam *et al. BMC Bioinformatics*　(2016) 17:435

Page 11 of 11

Moreover, the datasets supporting the conclusions of this article are available in the BIGMAC repository (by issuing python reproduce.py), and hyperlinks to datasets are located in

- https://berkeley.box.com/shared/static/ m3x4czbxbh87vq078li3llr3w0ba3i8n.tar
- https://berkeley.box.com/shared/static/ pq98sgh3lb6vqvxrsczfrphidf1ik1gj.tar
- https://berkeley.box.com/shared/static/ h05c4hl6fk8h8k95w0y17jtkeuzsz0kp.tar
- https://berkeley.box.com/shared/static/ 7ub5gqnd61pfvay6knl0ipklmrbzcsjq.tar

Authors' contributions
K-KL, RH, AC and SR contributed in the design of the study and writing of the manuscript. K-KL implemented the software. All authors read and approved the final manuscript.

Competing interests
Richard Hall is an employee of Pacific Biosciences at the time the manuscript is written.

Consent for publication
Not applicable.

Ethics approval and consent to participate
Not applicable.

Author details
[1]Department of Electrical Engineering and Computer Sciences, UC Berkeley, Berkeley, USA. [2]Pacific Biosciences, Menlo Park, USA. [3]Joint Genome Institute, Walnut Creek, USA.

References
1. Chen K, Pachter L. Bioinformatics for whole-genome shotgun sequencing of microbial communities. PLoS Comput Biol. 2005;1(2):106–12.
2. The Critical Assessment of Metagenome Interpretation (CAMI) competition. http://blogs.nature.com/methagora/2014/06/the-critical-assessment-of-metagenome-interpretation-cami-competition.html. Accessed 29 Mar 2016.
3. Koren S, Phillippy AM. One chromosome, one contig: complete microbial genomes from long-read sequencing and assembly. Curr Opin Microbiol. 2015;23:110–20.
4. Lam KK, LaButti K, Khalak A, Tse D. Finishersc: a repeat-aware tool for upgrading de novo assembly using long reads. Bioinformatics. 2015;31(19):3207–209.
5. Chin CS, Alexander DH, Marks P, Klammer AA, Drake J, Heiner C, Clum A, Copeland A, Huddleston J, Eichler EE, et al. Nonhybrid, finished microbial genome assemblies from long-read smrt sequencing data. Nat Methods. 2013;10(6):563–9.
6. Gurevich A, Saveliev V, Vyahhi N, Tesler G. Quast: quality assessment tool for genome assemblies. Bioinformatics. 2013;29(8):1072–1075.
7. Delcher AL, Phillippy A, Carlton J, Salzberg SL. Fast algorithms for large-scale genome alignment and comparison. Nucleic Acids Res. 2002;30(11):2478–483.
8. Koren S, Schatz MC, Walenz BP, Martin J, Howard JT, Ganapathy G, Wang Z, Rasko DA, McCombie WR, Jarvis ED, et al. Hybrid error correction and de novo assembly of single-molecule sequencing reads. Nat Biotechnol. 2012;30(7):693–700.
9. Przyborowski J, Wilenski H. Homogeneity of results in testing samples from poisson series: With an application to testing clover seed for dodder. Biometrika. 1940;31(3/4):313–323. JSTOR.
10. Myers EW. The fragment assembly string graph. Bioinformatics. 2005;21(suppl 2):79–85.
11. Elias I. Settling the intractability of multiple alignment. J Comput Biol. 2006;13(7):1323–1339.
12. Documentation of MetaFinisherSC. https://github.com/kakitone/ MetaFinisherSC. Accessed 29 Mar 2016.
13. Hall RJ, Chin CS, Mehrotra S, Juretic N, Wasserscheid J, Dewar K. An interactive workflow for the analysis of contigs from the metagenomic shotgun assembly of SMRT Sequencing data. 2014. http://files.pacb.com/ pdf/RHall_ASM2014_InteractiveWorkflow.pdf. Accessed 29 Mar 2016.
14. PacBio. PacBio Devnet. https://github.com/PacificBiosciences/DevNet/ wiki/Human_Microbiome_Project_MockB_Shotgun. Accessed 29 Mar 2016.
15. Boetzer M, Pirovano W. Sspace-longread: scaffolding bacterial draft genomes using long read sequence information. BMC Bioinforma. 2014;15(1):1.
16. English AC, Richards S, Han Y, Wang M, Vee V, Qu J, Qin X, Muzny DM, Reid JG, Worley KC, et al. Mind the gap: upgrading genomes with pacific biosciences rs long-read sequencing technology. PloS ONE. 2012;7(11): 47768.