Bioinformatics, 33(2), 2017, 169–176
doi: 10.1093/bioinformatics/btw597
Advance Access Publication Date: 14 September 2016
Original Paper

Genome analysis

BOSS: a novel scaffolding algorithm based on an optimized scaffold graph

Junwei Luo[1,2], Jianxin Wang[1,*], Zhen Zhang[1], Min Li[1] and Fang-Xiang Wu[3]

[1]School of Information Science and Engineering, Central South University, ChangSha 410083, China, [2]College of Computer Science and Technology, Henan Polytechnic University, JiaoZuo 454000, China and [3]Division of Biomedical Engineering, University of Saskatchewan, Saskatchewan S7N 5A9, Canada

*To whom correspondence should be addressed.

Associate Editor: John Hancock

Received on April 18, 2016; revised on June 22, 2016; accepted on September 8, 2016

Abstract

Motivation: While aiming to determine orientations and orders of fragmented contigs, scaffolding is an essential step of assembly pipelines and can make assembly results more complete. Most existing scaffolding tools adopt scaffold graph approaches. However, due to repetitive regions in genome, sequencing errors and uneven sequencing depth, constructing an accurate scaffold graph is still a challenge task.

Results: In this paper, we present a novel algorithm (called BOSS), which employs paired reads for scaffolding. To construct a scaffold graph, BOSS utilizes the distribution of insert size to decide whether an edge between two vertices (contigs) should be added and how an edge should be weighed. Moreover, BOSS adopts an iterative strategy to detect spurious edges whose removal can guarantee no contradictions in the scaffold graph. Based on the scaffold graph constructed, BOSS employs a heuristic algorithm to sort vertices (contigs) and then generates scaffolds. The experimental results demonstrate that BOSS produces more satisfactory scaffolds, compared with other popular scaffolding tools on real sequencing data of four genomes.

Availability and Implementation: BOSS is publicly available for download at https://github.com/bioinfomaticsCSU/BOSS.

Contact: jxwang@mail.csu.edu.cn

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

1 Introduction

As the increasing availability of next-generation sequencing technology, many de novo genome assemblers were developed for reconstructing complete and correct genome sequences (Gnerre *et al.*, 2011; He *et al.*, 2013; Luo *et al.*, 2015a). Genome assemblers usually first produce a large number of fragmented contigs. Then, scaffolding in the pipeline of assembly takes the contigs and paired reads as input to produce some scaffolds (Li *et al.*, 2016). A scaffold consists of oriented and ordered contigs. These scaffolds could benefit downstream analysis such as gene order, comparative or functional genomics and patterns of recombination (Hunt *et al.*, 2014).

Most existing scaffolding tools use scaffold graphs as the foundation for scaffolding. In a scaffold graph, a vertex represents a contig, an edge between two vertices represents the relative orientations and orders of them. When constructing a scaffold graph, there are two ways to decide whether an edge should be added between two vertices. One common way is based on the number of paired reads that link two contigs, if that number is larger than a threshold (defined by scaffolding tools or users), an edge is added and the edge weight is equal to that number. Another way is based on the distribution of paired reads that link two vertices. BESST (Sahlin *et al.*, 2014) uses the standard deviation of gap distance and read position

distribution to decide whether an edge between two vertices should be added.

Due to repetitive regions in genome, sequencing errors and uneven sequencing depth, two non-adjacent contigs possibly have paired reads that link them. As a result some spurious edges, which increases the difficulty of scaffolding, may be introduced in a scaffold graph. The spurious edges usually cause some contradictions among the orientations or orders of contigs which are derived from different edges (Bodily *et al.*, 2016). There are two ways to avoid the negative effect of spurious edges. One way adopts heuristic algorithms to select paths from the scaffold graph that maximize the weight of edges on the paths. SSPACE (Boetzer *et al.*, 2011) uses a greedy heuristic algorithm to select paths. OPERA (Gao *et al.*, 2011) utilizes the dynamic programming to choose paths. BESST (Sahlin *et al.*, 2014) adopts the breadth first search to pick up the path with the maximum weights of edges connecting two larger contigs. ScaffMatch (Mandric and Zelikovsky, 2015) proposed a novel optimization formulation representing scaffolding as a maximum-weight acyclic 2-matching problem. Another way transforms the detection of spurious edges as finding a set of edges that minimize the weight of edges and whose removal can lead to no contradictions in the scaffold graph. SCARPA (Donmez and Brudno, 2013) and SILP2 (Lindsay, 2014) adopt the linear programming to remove spurious edges. MIP (Salmela *et al.*, 2011) segments the scaffold graph into small sub-graphs, and uses the mixed integer programming to detect and remove spurious edges.

There are two problems which prevent most current scaffolding tools from getting more accurate results. (1) If the weight of an edge is equal to the number of paired reads, the edges among vertices which come from low depth sequencing regions are possibly regarded as spurious. Using the distribution of paired reads can partly resolve problems caused by repetitive regions and uneven sequencing depth. To our knowledge, BESST (Sahlin *et al.*, 2014) is the only scaffolding tool using the distribution of paired reads for constructing a scaffold graph, but BESST only considers paired reads that link two contigs to analyze the distribution and still sets the weight to be the number of paired reads. Yet, the paired reads whose only one mate read is mapped to contigs are also helpful to analyze the reliability of edges. (2) Existing methods prefer to consider all the edges simultaneously to detect spurious edges in scaffold graphs. However, the high weight edges are more possibly correct. It is appealing to develop a new method to take advantage of orientation and order information about high weight edges to detect the spurious edges.

In this paper, we utilize two new ideas to address these two problems. To address problem (1), for two contigs which have paired reads that link them, if one read is mapped to one contig, then we can infer the probability that its mate read can be mapped to another contig based on the distribution of *insert size*. Based on this idea, for two contigs, we consider all reads mapped to one contig no matter whether their mate reads can be mapped to another contig or not, and develop a statistical method to compute the expectation number of paired reads that link them. Then, the expectation number is compared with the real number to get a score which is used to decide whether an edge should be added or not, and the score is set to be the edge weight. In case that two adjacent contigs come from low depth sequencing regions, the edge weight still has a chance to be high by this method. If two non-adjacent contigs have paired reads that link them caused by repetitive regions, the expectation number and the real number of paired reads commonly differ greatly. Therefore, our statistical method can more accurately judge whether an edge between two vertices should be added and how the edge should be weighed. To address problem (2), we adopt an iterative strategy to detect and remove spurious edges. In the first iteration, we extract a sub-graph from the scaffold graph which is induced from edges with high weight, then we iteratively add the rest of edges to the sub-graph from high to low weight. In each iteration, we detect and remove spurious edges based on the sub-graph. The edges which have been confirmed to be non-spurious are used as prior information to guide the removal in subsequent iterations. The orientation and order information about high weight edges can be utilized in this iterative strategy.

Based on these two new ideas, we present a novel scaffolder BOSS (Building Optimized Scaffold graph for Scaffolding) to determine orientations and orders of contigs. BOSS is compared with other popular scaffolding tools on four real datasets, the experimental results demonstrate BOSS can generate more satisfactory scaffolds.

2 Notation

In this paper, a paired read *pr* is referred as two reads with opposite orientations which are sequenced from two ends of a particular fragment, *lr*(*pr*) is the left read of *pr* while *rr*(*pr*) is the right read of *pr*. The length of the fragment is called the *insert size* which approximately follows a normal distribution $N(\mu_{is}, \sigma_{is})$ (Luo *et al.*, 2015b). If a read *r* is forwardly mapped to a contig c_i, $crd(r,c_i)$ is the distance between the start mapping position of *r* and 3′-end of c_i. If the read *r* is reversely mapped to c_i, $crd(r,c_i)$ is the distance between the start mapping position of *r* and 5′-end of c_i. The gap distance between c_i and c_j is gd_{ij}. For a paired read *pr* that links c_i and c_j, we can get the $crd(lr(pr),c_i)$ and $crd(rr(pr),c_j)$ (shown in Fig. 1).

3 Methods

BOSS uses one or more paired read libraries for scaffolding. Each paired read library is stored in two FASTQ files. Before scaffolding, a set of contigs should be generated by an assembler. Each paired read library is mapped to the set of contigs, and the mapping information is stored in a BAM file. Then, BOSS takes the set of contigs and BAM files as input. The procedure of BOSS is outlined as follows: (i) Preprocessing; BOSS first analyzes the BAM files and filters out some ambiguous mapping information. (ii) Constructing an optimized scaffold graph; BOSS utilizes a statistical method to calculate the expectation number of paired reads that link two contigs. Based on the ratio of the expectation number to the real number, BOSS decides whether an edge should be added and how the edge should be weighed. Furthermore, BOSS uses an iterative strategy to detect and remove spurious edges. (iii) Sorting vertices in the scaffold graph; BOSS adopts a heuristic algorithm to sort vertices thus generating scaffolds.

3.1 Preprocessing

Due to repetitive regions and sequencing errors, a read may have multiple mapping positions. BOSS only keeps the read mapping

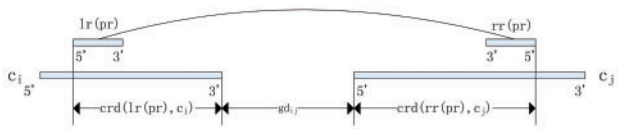

Fig. 1. For one paired read *pr* and two contigs c_i and c_j, *lr*(*pr*) is forwardly mapped to c_i, *rr*(*pr*) is reversely mapped to c_j

position with the highest score and discards the rest. Because the *insert size* approximately follows a normal distribution $N(\mu_{is}, \sigma_{is})$, the probability that the *insert size* is beyond the range $[\mu_{is} - 3 * \sigma_{is}, \mu_{is} + 3 * \sigma_{is}]$ is less than 5%. For a paired read mapped to the same contig, if its *insert size* is beyond $[\mu_{is} - 3 * \delta_{is}, \mu_{is} + 3 * \sigma_{is}]$, it is considered as abnormal and its mapping information will be removed. For a paired read *pr* that links two contigs c_i and c_j, if the sum of $crd(lr(pr),c_i)$ and $crd(rr(pr),c_j)$ is greater than $\mu_{is} + 3 * \sigma_{is}$, its mapping information will also be removed. Sequencing errors commonly cause that some reads cannot be mapped to any position. BOSS calculates the probability, α, that a read can be mapped to a contig, which is the ratio of the number of mapped reads left to the total number of reads. Next, BOSS calculates read coverage for each position of contigs based on read mapping information, then BOSS computes, σ_{rc}, the standard deviation of the read coverage and the average of read coverage. BOSS removes paired reads whose one mate read is mapped to the region whose read coverage is $2*\sigma_{rc}$ larger than the average.

3.2 Constructing a scaffold graph

In this step, BOSS constructs a scaffold graph G with the vertex set V and the edge set E. A vertex v_i represents a contig c_i. An edge e_{ij} is represented by a six-tuple $(v_i, v_j, o_i, o_j, gd_{ij}, w_{ij})$, where v_i and v_j are two vertices, o_i and o_j are mapping orientations of paired reads to v_i and v_j, respectively, gd_{ij} is the gap distance between v_i and v_j and w_{ij} is the weight of the edge.

3.2.1 Adding edges between vertices

If there exists paired reads that link two vertices v_i and v_j (c_i and c_j), BOSS uses the statistical method to decide whether an edge should be added between them and how the edge should be weighed. The statistical method includes five steps.

(1) Determining the mapping orientations between vi and vj. If the paired reads that link v_i and v_j have two or more different mapping orientation pairs (o_i, o_j), BOSS only keeps the mapping orientation pair which is supported by the larger number of paired reads and discards paired reads with other mapping orientation pairs. After that, if the number of paired reads that link them is smaller than a threshold num_{min} (default 2), a parameter in BOSS, an edge will not be added between them and the statistical method is terminated.

(2) Computing the gap distance between vi and vj. For the *t*th paired read pr_t that links v_i and v_j, BOSS calculates $crd(lr(pr),v_i)$ and $crd(rr(pr),v_j)$) (an example in Fig. 1). Based on all paired reads that link them, BOSS first calculates gd_{ij} by the following formula:

$$gd_{ij} = \frac{1}{n}\sum_{t=1}^{n}(\mu_{is} - crd(lr(pr_t), v_i) - crd(rr(pr_t), v_j)) \quad (1)$$

where n is the number of paired reads that link v_i and v_j, pr_t is the *t-th* paired read.

(3) Computing the expectation number of paired reads between vi and vj based on reads mapped to vi. Given a read r which has been mapped to v_i (c_i) and its mapping orientation is the same as o_i, if the mate read of r can be mapped to v_j, $l_{\min}$ and $l_{\max}$ are the smallest and largest distance between them. $l_{\min}$ is the sum of $crd(r,v_i)$, gd_{ij} and $len(r)$. $len(r)$ is the length of r. $l_{\max}$ is the sum of $crd(r,v_i)$,

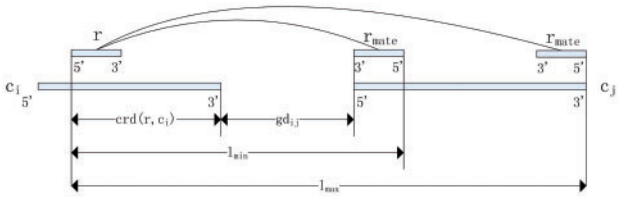

Fig. 2. When the read r has been mapped to c_i, the probability that its mate read r_{mate} can be mapped to c_j is the probability that their *insert size* falls in $[l_{\min}, l_{\max}]$. When r_{mate} is mapped to the most left end of c_j, $l_{\min}$ is the smallest *insert size*. When r_{mate} is mapped to the most right end of c_j, $l_{\max}$ is the largest *insert size*

gd_{ij} and $len(v_j)$. $len(v_j)$ is the length of v_j (shown in Fig. 2). The probability $p(r,v_i,v_j)$ that its mate read can be mapped to v_j (c_j) is the probability that the *insert size* falls in the interval $[l_{\min}, l_{\max}]$, which can be calculated by the following formula:

$$p(r, v_i, v_j) = \alpha * \int_{l_{\min}}^{l_{\max}} f(x)dx \quad (2)$$

where α is the probability that one read can be mapped to a contig, which has been acquired in the step of preprocessing. $f(x)$ is the probability density function of $N(\mu_{is}, \sigma_{is})$.

Finally, the expectation number, $\exp_{ij}$, of paired reads that link the two vertices is calculated by the following formula:

$$\exp_{ij} = \sum_{r \in RS} p(r, v_i, v_j) \quad (3)$$

where RS is the set of reads. Because the insert size larger than $\mu_{is} + 3 * \sigma_{is}$ is regarded as abnormal, RS only consists of reads which fall in the region $[\max(\mu_{is} + 3 * \sigma_{is} - gd_{ij} - len(r), 0), len(v_i)]$ of v_i and whose mapping orientations are the same as o_i, no mater whether their mate reads are mapped to v_j or not. Then, BOSS defines:

$$\rho_{ij} = \min\left(\frac{\exp_{ij}}{n}, \frac{n}{\exp_{ij}}\right) \quad (4)$$

where ρ_{ij} is to measure how close the expectation number of paired reads is to the real number.

(4) Computing the expectation number of paired reads between vi and vj based on reads mapped to vj. BOSS can also get another expectation number $\exp_{ji}$ based on reads mapped to v_j in the same way described in the previous step, and get ρ_{ji}.

(5) Computing the edge weight. If the arithmetic mean of ρ_{ij} and ρ_{ji} are greater than a threshold $w_{\min}$ (default 0.2), BOSS adds an edge e_{ij} between v_i and v_j. If o_i (o_j) is a forward mapping, the edge e_{ij} connects 3′-end of v_i (v_j), else 5′-end. w_{ij} is equal to the arithmetic mean of ρ_{ij} and ρ_{ji}. If ρ_{ij} or ρ_{ji} is smaller than $w_{\min}$, but v_i has no paired reads that link other vertices with the same o_i, so is v_j, BOSS also adds an edge between them and w_{ij} is set to be $w_{\min}$. When computing the edge weight, BOSS can use not only the arithmetic mean but also the geometric mean, which is controlled by a parameter in BOSS. The influence of arithmetic mean and geometric mean on BOSS is shown in Supplementary Material. BOSS adopts arithmetic mean in default.

After processing all paired contigs, a weighted scaffold graph is constructed. If the length of a vertex is equal to or larger than $\mu_{is} + 3 * \sigma_{is}$, the vertex is defined as a long vertex. For a long vertex

and its edges connecting other long vertices from the same end, BOSS only keeps the edge with the maximum weight.

3.2.2 Detecting spurious edges

Spurious edges are usually caused by complex repetitive regions which are difficult to judge by using the local paired reads that link two vertices. BOSS performs the detection of spurious edges by iteratively assigning the orientations and the orders of vertices, and maximizing the weights of non-spurious edges. An edge is non-spurious if the orientation and the order information provided by it are consistent with the assignment, else spurious.

(1) Detecting spurious edges by assigning the orientations of vertices. In this step, BOSS tries to find the optimal assignment of orientations that maximize the sum weight of non-spurious edges by an integer linear programming model *LP*(*OB*, *CS*) (see Algorithm 1). *OB* is the objective function and *CS* is the set of constraints. $s_i \in \{0, 1\}$ denotes orientation of v_i. $\eta_{ij} \in \{0, 1\}$ represents whether e_{ij} is spurious or not. BOSS iterates w from $w_{\max}$ to $w_{\min}$ with a step of 0.1, w is an edge weight threshold used to construct a sub-graph based on the scaffold graph. In the first iteration, $w = w_{\max}$, BOSS constructs a sub-graph G_s which consists of edges whose weights are in $[w,1)$. In the next iteration, w is set to w - 0.1, BOSS adds edges whose weights are in $[w, w + 0.1)$ to G_s. In each iteration, BOSS constructs *LP*(*OB*, *CS*) and solves it. After getting the orientation assignment, BOSS removes spurious edges, while the remaining edges are non-spurious and are used as prior information to guide detection of spurious edges in the subsequent iterations. The iteration will be terminated after w is smaller than $w_{\min}$.

In each iteration, the orientation of each vertex in G_s is a variable. *TE* is a set of edges which are added to G_s in this iteration. For an edge $e_{ij} \in G_s$, if $e_{ij} \notin TE$, and $o_i \neq o_j$, BOSS adds the following constraint equation (5) to *CS* in which $\eta_{ij} = 1$. If $e_{ij} \notin TE$, and $o_i = o_j$, BOSS adds the following constraint equation (6) to *CS* in which $\eta_{ij} = 1$. If $e_{ij} \in TE$, and $o_i \neq o_j$, BOSS adds the following constraint equation (5) to *CS* in which η_{ij} is a variable. If $e_{ij} \in TE$, and $o_i = o_j$, BOSS adds the following constraint equation (6) to *CS* in which η_{ij} is a variable.

$$\eta_{ij} \leq s_i + s_j \leq 2 - \eta_{ij} \tag{5}$$

$$\eta_{ij} - 1 \leq s_i - s_j \leq 1 - \eta_{ij} \tag{6}$$

Then BOSS maximizes the sum weight of non-spurious edges, and the objective function is:

$$OB = \mathrm{MAX}\left(\sum_{e_{ij} \in TE} (w_{ij} * \eta_{ij})\right) \tag{7}$$

BOSS uses the branch-and-bound algorithm to solve *LP*(*OB*, *CS*) and gets the approximate optimal assignment. If $\eta_{ij} = 0$, e_{ij} is spurious and BOSS removes e_{ij} from G_s. If $\eta_{ij} = 1$, e_{ij} is non-spurious. After all iterations, the scaffold graph G is replaced with G_s. Finally BOSS reverses and complements some vertices to make sure that two ends of each edge correspond to *5′-end* of one vertex and *3′-end* of another vertex, respectively.

(2) Detecting spurious edges by assigning the coordinates of vertices. In this step, BOSS assigns a starting coordinate for each vertex and tries to find the best assignment such that the gap distances between vertices calculated by starting coordinates agree the best with the gap distances suggested by edges. BOSS still uses the iterative strategy to construct sub-graph G_s as the above process. In each iteration, BOSS constructs a new *LP*(*OB*, *CS*) based on G_s. *TE* is a set of edges which are added to G_s in this iteration. For G_s, if edge $e_{ij} \in TE$ which connects 3′-end of v_i and 5′-end of v_j, BOSS adds the following constraint to *CS* (Donmez and Brudno, 2013):

$$L(\phi_{ij} - 1) <= x_j - x_i - len(c_i) - gd_{ij} <= L(1 - \phi_{ij}) \tag{8}$$

$x_i \in [0, L)$ is an integer and denotes the staring coordinate of vertex v_i. L is a large constant and equal to twice the sum of the length of all contigs. ϕ_{ij} is a slack variable in the range $[0, 1]$ which reflects the consistency between two gap distances suggested by starting coordinates (x_i and x_j) and edge e_{ij}. If edge $e_{ij} \notin TE$, e_{ij} is non-spurious, and BOSS adds the following constraint to *CS*, which is used as prior information to identify spurious edges from *TE*.

$$0 < x_j - x_i - len(c_i) \leq \mu_{is} + 3 * \sigma_{is} \tag{9}$$

The objective function is:

$$OB = \mathrm{MAX}\left(\sum_{e_{ij} \in ES} (w_{ij} * \phi_{ij})\right) \tag{10}$$

After BOSS gets the approximate optimal assignment, for an edge in *TE*, if the gap distance suggested by starting coordinates is far away from the one suggested by the edge, this edge is spurious and BOSS removes it. After all iterations, the scaffold graph G is

Algorithm 1. Detect_orientation(G)

```
Initialization G_s = ∅; TE = ∅;w = w_max;CS = ∅
while w >= w_min do
  for each edge e_ij ∈ G do
    if w <= w_ij < w + 0.1 then
      add eij to Gs and TE
    end if
  end for
  for each edge e_ij ∈ G_s do
    if e_ij ∉ TE then
      if o_i ≠ o_j then
        add "s_i + s_j = 1"to CS
      else
        add "s_i = s_j"to CS
      end if
    else
      if o_i ≠ o_j then
        add"η_ij <= s_i + s_j <= 2 − η_ij" to CS
      else
        add "η_ij − 1 <= s_i − s_j <= 1 − η_ij" to CS
      end if
    end if
  end for
  OB = MAX(∑_{e_ij∈TE}(w_ij * η_ij)) and solve LP(OB, CS)
  for each edge e_ij ∈ TE do
    if ηij == 0then
      delete eij from Gs
    end if
  end for
  TE = ∅, CC = ∅and w = w − 0.1
end while
replace G with Gs and output G
```

replaced with G_s, and BOSS gets the starting coordinate of each vertex. If two vertices occupy the same coordinates, BOSS traverses the scaffold graph from the two vertices separately to find the first vertex they meet, and removes the edge with the smaller weight entering the first vertex.

3.3 Sorting vertices in scaffold graph

In this step, BOSS produces scaffolds by sorting vertices in the scaffold graph G. BOSS first extracts simple paths only including long vertices whose lengths are equal to or larger than $\mu_{is} + 3 * \sigma_{is}$, and these simple paths compose a path set PS. Second, BOSS identifies short vertices (whose lengths are smaller than $\mu_{is} + 3 * \sigma_{is}$) which have edges connecting with two adjacent long vertices in the same path, then BOSS inserts these short vertices into the middle of the two adjacent long vertices. Finally, for a path, BOSS extends it through short vertices which have not been inserted to the paths in PS. In the extending process, BOSS adopts the breadth-first search to sort the short vertices based on the scaffold graph, gap distances and contig lengths, these sorted vertices are appended to the paths. When the extension meets one end of another path, these two paths are merged. The process is terminated if there are no vertices that have not been visited. An illustrative example is shown in Figure 3.

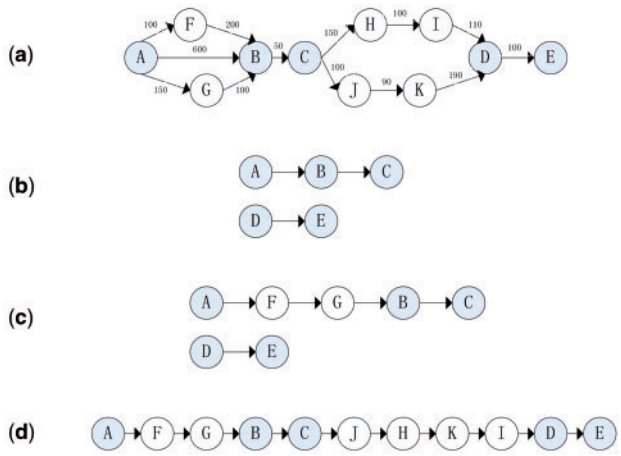

Fig. 3. (**a**) A scaffold graph, each edge is directed from 3′-end of a vertex to 5′-end of another vertex. A, B, C, D and E are long vertices, the rest are short vertices, the numbers on edges are gap distances between vertices. (**b**) ABC and DE are two simple paths of long vertices, the path set PS = {ABC, DE}. (**c**) Short vertices F and G both have edges connected to A and B, then F and G are inserted in path ABC, the orders of F and G are determined based on the gap distances between vertices, PS = {AFGBC, DE}. (**d**) Extending path AFGBC, C is considered as a starting vertex and BOSS uses the breadth-first search to sort short vertices based on the distance from C to themselves. When facing D, the extension will terminates and two paths are merged to one new path, PS = {AFGBCJHKIDE}

4 Experiment

4.1 Datasets

To evaluate the performance of BOSS, experiments are carried out on four real datasets used in Hunt *et al.* (2014). These datasets include Illumina reads from the *Staphylococcus aureus* (*S.aureus*), *Rhodobacter sphaeroides* (*R.sphaeroides*), Human chromosome 14 and the *Plasmodium falciparum* (*P.falciparum*) clone 3D7 reference genome. Details about four datasets are listed in Table 1. The mapping read ratio is calculated based on Bowtie2 (Langmead and Salzberg, 2012). The first two datasets include only one read library and the last two datasets contain two read libraries with different properties. Contig sets are generated by genome assembler Velvet (Zerbino and Birney, 2008).

4.2 Evaluation metrics

Although some users might think that good scaffolds should have a large N50 and CN50 (Salzberg *et al.*, 2012), these metrics do not necessarily reflect correctly orientated or ordered contigs in a scaffold (Hunt *et al.*, 2014; Mandric and Zelikovsky, 2015). For assessing the quality of scaffolds reliably, a novel evaluation tool is proposed by Hunt *et al.* (2014) in which each contig is represented by a sequence tag and four key metrics are presented: (1) Correct Joins (CJ). (2) Incorrect Joins (IJ). (3) Skipped Tags (ST). (4) Lost Tags (LT). The last three metrics are bad joins. We set the weights of four metrics of CJ, IJ, LT and ST as 1, 1, 2 and 0.5, respectively, according to the widely accepted rule (Hunt *et al.*, 2014). The number of potential joins is the number of contigs that can be joined to the scaffold which is the number of contigs minus the number of chromosomes. After we get the correct joins (CJ) and bad joins (IJ, ST and LT) of scaffolds, we adopt the *F-score* as a comprehensive metric which is calculated by *TPR* and *PPV* (Mandric and Zelikovsky, 2015).

4.3 Statistical method and iterative strategy

For verifying the effectiveness of the statistical method and the iterative strategy presented in this paper, we compare other two different versions of BOSS: $BOSS_1$ and $BOSS_2$. The difference among $BOSS_1$, $BOSS_2$ and BOSS is that they adopt different strategies for constructing a scaffold graph. When constructing a scaffold graph, $BOSS_1$ adds an edge between two vertices if the number of paired reads that links them is larger than $num_{\min}$. $BOSS_2$ uses the statistical method (described in section 3.2.1) for constructing a scaffold graph. BOSS not only adopts the statistical method but also the iterative strategy to construct a scaffold graph. The scaffolding results about *S.aureus* and *P.falciparum* are shown in Table 2. The scaffolding results about *R.sphaeroides* and Human chromosome 14 are shown in Table 3.

From Tables 2 and 3, we can find that the *F-score* of $BOSS_2$ is larger than that of $BOSS_1$, and the *F-score* of BOSS is larger than that of $BOSS_2$ for dataset of *S.aureus*, dataset of *R.sphaeroides*,

Table 1. Details of real datasets

	Staphylococcus aureus	*Rhodobacter sphaeroides*	*Plasmodium falciparum*		Human chromosome 14	
Genome size(Mbp)	2.9	4.6	23.3		88.2	
Read length(bp)	37	101	76	75	101	57
Number of reads(M)	3.5	2.1	52.5	12.0	22.7	2.4
Coverage	~45	~46	~171	~39	~26	~2
Insert size(bp)	3500	3700	650	2700	2900	34500
Mapping read ratio	0.548	0.693	0.778	0.426	0.729	0.890

Table 2. Performances of $BOSS_1$, $BOSS_2$ and BOSS on *S.aureus* and *P.falciparum*

	Dataset of *S.aureus*			Short insert size Dataset of *P.falciparum*			Long insert size Dataset of *P.falciparum*			Short and long insert size Datasets of *P.falciparum*		
	TPR	PPV	F-score	TPR	PPV	F-score	TPR	PPV	F-score	TPR	PPV	F-score
$BOSS_1$	0.725	0.569	0.638	0.585	0.737	0.652	0.747	0.648	0.694	0.820	0.751	0.784
$BOSS_2$	0.886	0.851	0.868	0.634	0.896	0.742	0.828	0.813	0.821	0.911	0.910	0.910
BOSS	0.862	0.878	0.870	0.642	0.921	0.757	0.798	0.816	0.807	0.920	0.928	0.924

Table 3. Performances of $BOSS_1$, $BOSS_2$ and BOSS on *R.sphaeroides* and Human chromosome 14

	Dataset of *R.sphaeroides*			Short insert size Dataset of Human			Long insert size Dataset of Human			Short and long insert size Datasets of Human		
	TPR	PPV	F-score	TPR	PPV	F-score	TPR	PPV	F-score	TPR	PPV	F-score
$BOSS_1$	0.751	0.638	0.690	0.661	0.528	0.587	0.305	0.452	0.364	0.688	0.532	0.600
$BOSS_2$	0.863	0.801	0.831	0.789	0.692	0.738	0.239	0.556	0.334	0.806	0.697	0.748
BOSS	0.870	0.881	0.876	0.766	0.783	0.775	0.233	0.559	0.329	0.767	0.745	0.756

Table 4. Evaluation results about *S.aureus* and *R.sphaeroides*

	S.aureus					*R.sphaeroides*				
	N50	CN50	TPR	PPV	F-score	N50	CN50	TPR	PPV	F-score
ABySS	619764	619764	0.593	0.904	0.716	280984	276804	0.674	0.904	0.772
Bambus2	242814	242650	0.569	0.852	0.682	146002	145952	0.577	0.896	0.702
MIP	–	–	–	–	–	488095	487941	0.735	0.823	0.777
OPERA	1084108	**686577**	0.671	0.772	0.718	108182	108172	0.554	0.959	0.703
SCARPA	112264	112083	0.461	0.675	0.548	37667	37581	0.367	0.907	0.522
SGA	309286	309153	0.497	0.922	0.646	42825	42722	0.407	0.939	0.568
SOAP2	643384	621109	0.784	0.811	0.798	2522483	2522482	0.821	0.942	**0.877**
SOPRA	112278	112083	0.240	0.842	0.373	32232	30492	0.425	0.852	0.567
SSPACE	332784	261710	0.629	0.764	0.690	109776	108410	0.626	0.903	0.740
BESST	1716351	335064	0.671	0.775	0.719	1021151	1020921	0.846	0.896	0.870
ScaffMatch	1476925	351546	0.832	0.779	0.805	2547006	2528248	0.644	0.970	0.774
BOSS	1035497	596371	0.862	0.878	**0.870**	2548917	**2546780**	0.870	0.881	0.876

short insert size dataset of *P.falciparum*, short and long insert size datasets of *P.falciparum*, short insert size dataset of Human chromosome 14, and short and long insert size datasets of Human chromosome 14. For the long insert size dataset of *P.falciparum*, although the *F-score* of $BOSS_2$ is larger than that of $BOSS_1$, the *F-score* of BOSS is smaller than that of $BOSS_2$. For the long insert size dataset of Human chromosome 14, the *F-score* of $BOSS_1$ is the largest and the *F-score* of BOSS is smaller than that of $BOSS_2$. In conclusion, the statistical method and the iterative strategy are effective for most cases. Below we discuss the case that BOSS produces more satisfactory scaffolding results.

4.4 Comparisons of scaffolders and discussion

We compare BOSS with popular scaffolders Bambus2 (Koren *et al.*, 2011), MIP (Salmela *et al.*, 2011), OPERA (Gao *et al.*, 2011), SCARPA (Donmez and Brudno, 2013), SOPRA (Dayarian *et al.*, 2010) and SSPACE (Boetzer *et al.*, 2011), and scaffolding modules from assemblers ABYSS (Simpson *et al.*, 2009), SOAP2 (Luo *et al.*, 2012) and SGA (Simpson and Durbin, 2012). We compute the *F-score* of these scaffolders based on scaffolding results which are abstracted from Hunt *et al.* (2014) whose datasets are used in this paper. Moreover, two recently published scaffolders BESST (Sahlin *et al.*, 2014) and ScaffMatch (Mandric and Zelikovsky, 2015) are also considered, and we compute the *F-score* of these two scaffolders based on Mandric and Zelikovsky (2015) who adopt the same datasets.

Some scaffolders contain the mapping process by one or more mapping tools in their own pipeline, others need SAM or BAM files produced by users themselves. Read mapping information produced by different mapping tools usually leads to distinct scaffolding results. For conducting unbiased investigation, three mapping tools: Bowtie (Langmead *et al.*, 2009), Bowtie2 (Langmead and Salzberg, 2012) and BWA (Li and Durbin, 2009) are used. Bowtie maps any read which has no mismatches (when using the option -v 0) or up to three mismatches (when using -v 3). We only show the evaluation results of scaffolders using Bowtie2 except scaffolders containing their own mapping tools. All evaluation results, and the running time and peak memory of BOSS are provided in Supplementary Materials.

For *S.aureus*, there are 170 contigs and 167 potential joins, and the scaffolding results are shown in Table 4. For *R.sphaeroides*, there are 577 contigs and 570 potential joins, and the scaffolding

Table 5. Evaluation results about *P.falciparum*

	Short insert size					Long insert size				
	N50	CN50	TPR	PPV	F-score	N50	CN50	TPR	PPV	F-score
ABySS	5862	5689	0.590	0.973	0.734	3734	3548	0.247	0.882	0.385
Bambus2	3193	3093	0.307	0.940	0.463	29705	24913	0.548	0.825	0.659
MIP	6158	5485	0.596	0.876	0.710	88297	78672	0.834	0.737	0.782
OPERA	5035	4824	0.398	0.899	0.552	44667	40170	0.673	0.879	0.762
SCARPA	4912	4628	0.519	0.872	0.651	14037	9708	0.525	0.880	0.658
SGA	5324	5104	0.531	0.972	0.687	4438	4096	0.312	0.898	0.463
SOAP2	6234	5981	0.596	0.967	0.737	167570	**83851**	0.823	0.874	**0.848**
SOPRA	4954	4632	0.526	0.931	0.672	49671	44158	0.779	0.913	0.841
SSPACE	6011	5845	0.618	0.957	0.751	17796	15553	0.496	0.875	0.633
BESST	7471	3931	0.283	0.732	0.408	4133	2813	0.141	0.836	0.241
ScaffMatch	8626	5872	0.607	0.905	0.727	41564	25380	0.749	0.833	0.789
BOSS	7308	**6723**	0.642	0.921	**0.757**	76831	41075	0.798	0.816	0.807

Table 6. Evaluation results about Human chromosome 14

	Short insert size					Long insert size				
	N50	CN50	TPR	PPV	F-score	N50	CN50	TPR	PPV	F-score
ABySS	195177	191188	0.470	0.843	0.604	12262	12215	0.001	0.711	0.003
Bambus2	98162	85659	0.530	0.824	0.645	278682	72210	0.179	0.683	0.284
MIP	244064	**235731**	0.697	0.790	0.741	272440	49800	0.296	0.528	0.379
OPERA	214972	207047	0.617	0.877	0.724	73477	20677	0.185	0.554	0.277
SCARPA	58330	55760	0.499	0.886	0.638	43969	17786	0.080	0.667	0.144
SGA	134574	133192	0.490	0.858	0.623	–	–	–	–	–
SOAP2	282437	234561	0.790	0.889	**0.836**	220644	**86679**	0.227	0.669	0.338
SOPRA	100768	96436	0.740	0.909	0.816	79517	34750	0.147	0.641	0.240
SSPACE	78552	77361	0.480	0.867	0.618	77832	30449	0.138	0.676	0.229
BESST	146749	80218	0.400	0.816	0.537	13815	8828	0.006	0.621	0.012
ScaffMatch	131135	80329	0.623	0.847	0.718	148412	42523	0.298	0.858	**0.442**
BOSS	216675	132718	0.766	0.783	0.775	156553	43111	0.233	0.559	0.329

Table 7. Evaluation results about combination of two datasets

	P.falciparum					Human chromosome 14				
	N50	CN50	TPR	PPV	F-score	N50	CN50	TPR	PPV	F-score
ABySS	6828	6529	0.615	0.968	0.753	198501	195474	0.471	0.853	0.607
Bambus2	–	–	–	–	–	299753	99505	0.516	0.764	0.616
MIP	56672	38704	0.869	0.875	0.872	44372	31148	0.428	0.717	0.536
OPERA	42450	38409	0.692	0.873	0.772	1692782	**1062031**	0.645	0.876	0.743
SCARPA	36945	23951	0.789	0.846	0.816	134364	106654	0.537	0.875	0.666
SGA	6606	6134	0.528	0.943	0.677	134574	133192	0.490	0.858	0.624
SOAP2	12076	10629	0.643	0.911	0.754	561198	447849	0.790	0.885	**0.835**
SOPRA	16366	15511	0.754	0.972	0.849	112239	75046	0.523	0.830	0.641
SSPACE	6383	5982	0.633	0.954	0.761	66271	65222	0.464	0.868	0.605
BESST	25300	7621	0.422	0.755	0.542	295976	114434	0.416	0.826	0.553
ScaffMatch	78627	47662	0.884	0.875	0.879	802755	195239	0.635	0.866	0.733
BOSS	80036	**62896**	0.920	0.928	**0.924**	425575	135241	0.767	0.745	0.756

results are also shown in Table 4. For *P.falciparum*, there are 9318 contigs and 9302 potential joins. All scaffolders use two datasets for scaffolding separately, and the scaffolding results are shown in Table 5. For Human chromosome 14, there are 19936 contigs and 19935 potential joins, and the scaffolding results using two datasets separatively are shown in Table 6. For *P.falciparum* and Human chromosome 14, the scaffolding results using the combination of the short and long insert size datasets are shown in Table 7. '–' in Tables 4, 6 and 7 means that the results cannot be got by corresponding scaffolders.

In Table 4, 5, 6 and 7, the bold values represent the best values of CN50 or F-score. For eight datasets, BOSS has the best *F-score* for three datasets and the best *CN50* for three datasets. SOAP2 has the best *F-score* for four datasets and the best *CN50* for two datasets. OPERA has the best *CN50* for two datasets. ScaffMatch has the best *F-score* for one dataset. MIP has the best *CN50* for one dataset.

4.5 Discussion

By investigating and analyzing the scaffolding results of BOSS, we find that BOSS performs better on datasets whose ratio of the

number of mapping reads to the genome size is large. The ratio is symbolized as ω. We can calculate ω based on the read number, genome size and mapping read ratio which are shown in Table 1. For datasets of *S.aureus* and the short *insert size* dataset of *P.falciparum*, their values of ω are 0.661 and 1.753, respectively. BOSS gets the best *F-score* compared with all other scaffolders. We conjecture that BOSS produces more satisfactory scaffolding result when the value of ω for datasets is large.

For the dataset of *R.sphaeroides*, long *insert size* dataset of *Plasmodium falciparum*, short *insert size* dataset of Human chromosome 14, and long *insert size* dataset of Human chromosome 14, their values of ω are 0.316, 0.219, 0.188 and 0.024, and BOSS also can produce acceptable scaffolding results. The reason why the value of ω influences BOSS is that the paired reads between two actual adjacent contigs usually is large when the value of ω is large, and this is helpful in more precisely weighting edges based on the statistical method proposed in this paper.

In BOSS, the *insert size* and its standard deviation of the dataset are parameters provided by users. For further examining the impact of standard deviation on the scaffolding results, we conduct BOSS with different standard deviations based on the mapping tool Bowtie2. The scaffolding results are shown in Supplementary Material. From the results, we can see that different standard deviations influence the scaffolding results, but the range of F-score variation is not large. BOSS adopts $\sigma_{is} = 0.07 * \mu_{is}$ in default.

5 Conclusion

In this paper, we have presented a novel scaffolder BOSS for determining orientations and orders of contigs. BOSS employs a new statistical method to decide whether an edge between contigs should be added and how the edge should be weighed. In addition, BOSS adopts an iterative strategy to detect and remove spurious edges in the scaffold graph. Finally, BOSS sorts vertices in the scaffold graph, thus producing oriented and ordered vertices which correspond to scaffolds. The experiments have been conducted on four datasets. The results have illustrated that BOSS outperforms other competing scaffolders when the value of ω for datasets is large, and also can produce comparable scaffolding results in the other cases.

Funding

This work was supported in part by the National Natural Science Foundation of China under Grant No. 61232001, No. 61420106009, No. 61379108, No.61602156 and The National Science Fund for Excellent Young Scholars under Grant No. 61622213.

Conflict of Interest: none declared.

References

Bodily,P.M. *et al.* (2016) ScaffoldScaffolder: solving contig orientation via bidirected to directed graph reduction. *Bioinformatics*, **32**, 17–24.

Boetzer,M. *et al.* (2011) Scaffolding pre-assembled contigs using SSPACE. *Bioinformatics*, **27**, 578–579.

Dayarian,A. *et al.* (2010) SOPRA: scaffolding algorithm for paired reads via statistical optimization. *BMC Bioinformatics*, **11**, 345.

Donmez,N. and Brudno,M. (2013) SCARPA: scaffolding reads with practical algorithms. *Bioinformatics*, **29**, 428–434.

Gao,S. *et al.* (2011) Opera: reconstructing optimal genomic scaffolds with high-throughput paired-end sequences. *J. Comput. Biol.*, **18**, 1681–1691.

Gnerre,S. *et al.* (2011) High-quality draft assemblies of mammalian genomes from massively parallel sequence data. *Proc. Natl. Acad. Sci. U. S. A.*, **108**, 1513–1518.

He,Y. *et al.* (2013) De novo assembly methods for next generation sequencing data. *Tsinghua Sci. Technol.*, **5**, 500–514.

Hunt,M. *et al.* (2014) A comprehensive evaluation of assembly scaffolding tools. *Genome Biol.*, **15**, 42.

Koren,S. *et al.* (2011) Bambus 2: scaffolding metagenomes. *Bioinformatics*, **31**, 2964–2971.

Langmead,B. *et al.* (2009) Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biol.*, **10**, R25.

Langmead,B. and Salzberg,S. (2012) Fast gapped-read alignment with Bowtie 2. *Nat. Methods*, **9**, 357–359.

Li,H. and Durbin,R. (2009) Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics*, **25**, 1754–1760.

Li,M. *et al.* (2016) ISEA: iterative seed-extension algorithm for de novo assembly using paired-end information and insert size distribution. *IEEE/ACM Trans. Comput. Biol. Bioinf.*, doi: 10.1109/TCBB.2016.2550433.

Lindsay,J. *et al.* (2014) Ilp-based maximum likelihood genome scaffolding. *BMC Bioinformatics*, **15**, S9.

Luo,R. *et al.* (2012) SOAPdenovo2: an empirically improved memory-efficient short-read de novo assembler. *GigaScience*, **1**, 18.

Luo,J. *et al.* (2015a) EPGA: de novo assembly using the distributions of reads and insert size. *Bioinformatics*, **31**, 825–833.

Luo,J. *et al.* (2015b) EPGA2: memory-efficient de novo assembler. *Bioinformatics*, **31**, 3988–3990.

Mandric,I. and Zelikovsky,A. (2015) ScaffMatch: Scaffolding Algorithm Based on Maximum Weight Matching. *Bioinformatics*, **31**, 2632–2638.

Sahlin,K. *et al.* (2014) Besst-efficient scaffolding of large fragmented assemblies. *BMC Bioinformatics*, **15**, 281.

Salmela,L. *et al.* (2011) Fast scaffolding with small independent mixed integer programs. *Bioinformatics*, **27**, 3259–3265.

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

Salzberg,S. *et al.* (2012) GAGE: a critical evaluation of genome assemblies and assembly algorithms. *Genome Res.*, **22**, 557–567.

Simpson,J.T. and Durbin,R. (2012) Efficient de novo assembly of large genomes using compressed data structures. *Genome Res.*, **22**, 549C556.

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