ExSPAnder: a universal repeat resolver for DNA fragment assembly

Andrey D. Prijibelski1,†,*, Irina Vasilinetc1,†, Anton Bankevich1, Alexey Gurevich1, Tatiana Krivosheeva1, Sergey Nurk1, Son Pham4, Anton Korobeynikov1,2, Alla Lapidus1,3 and Pavel A. Pevzner1,4

1Algorithmic Biology Lab, St. Petersburg Academic University, St. Petersburg, Russia, 2Department of Mathematics and Mechanics, 3Theodosius Dobzhansky Center for Genome Bioinformatics, St. Petersburg State University, St. Petersburg, Russia and 4Department of Computer Science and Engineering, University of California, San Diego, USA

ABSTRACT
Next-generation sequencing (NGS) technologies have raised a challenging de novo genome assembly problem that is further amplified in recently emerged single-cell sequencing projects. While various NGS assemblers can use information from several libraries of read-pairs, most of them were originally developed for a single library and do not fully benefit from multiple libraries. Moreover, most assemblers assume uniform read coverage, condition that does not hold for single-cell projects where utilization of read-pairs is even more challenging. We have developed an exSPAnder algorithm that accurately resolves repeats in the case of both single and multiple libraries of read-pairs in both standard and single-cell assembly projects.

Availability and implementation: http://bioinf.spbau.ru/en/spades

Contact: ap@bioinf.spbau.ru

1 INTRODUCTION
Most of existing next-generation sequencing (NGS) platforms generate read-pairs—pairs of reads (called mates) that are sequenced from different ends of a genomic fragment with approximately known length (called the insert size). Because the insert size usually exceeds the length of a single read, read-pairs may match up unique regions surrounding repeats that are longer than the read length. A combination of several libraries of read-pairs with different insert sizes is often used to produce high-quality assemblies (Butler et al., 2008; Bresler et al., 2012). Paired-end libraries usually have insert size <1 kb and are used for resolving relatively short repeats. Jumping libraries are characterized by an average insert size of anywhere from 1 to 20 kb and are helpful in resolving longer repeats and contig scaffolding (inferring the order of contigs in the genome). However, because of a high insert size variation, information from jumping libraries is rather difficult to use for the purpose of assembly.

The problem of using multiple read-pair libraries was previously addressed by ALLPATHS-LG (Gnerre et al., 2011), Ray (Boisvert et al., 2010), Velvet (Zerbino and Birney, 2008) and some other assemblers. However, these tools are designed for standard (multiplex) assembles and do not perform well on single-cell datasets. On the other hand, the single-cell assemblers ESC (Chitsaz et al., 2011), IDBA-UD (Peng et al., 2012) and SPAdes (Bankevich et al., 2012) are designed for a single read-pair library. In addition, the recently proposed Paired de Bruijn Graph algorithms for repeat resolution (Medvedev et al., 2011; Pham et al., 2013; Vyahhi et al., 2012) also focus on a single library and it remains unclear how to extend them to multiple libraries.

We present exSPAnder algorithm that works with both single and multiple libraries in standard and single-cell assembly projects. exSPAnder uses a simple path extension approach for repeat resolution that was originally proposed in the Ray assembler (and later used in Telescopier (Bresler et al., 2012)) and combines it with some ideas from the Rectangle Graph approach (Bankevich et al., 2012; Vyahhi et al., 2012). Given a set of paths in the assembly graph (Bankevich et al., 2012) (i.e. simplified de Bruijn graph (Compeau et al., 2011; Pevzner et al., 2001) of k-mers in reads after removal of bulges, tips and chimeric edges), exSPAnder attempts to extend each path with the goal to generate longer paths. For a path P ending in a vertex v, we consider all edges starting at v (referred to as extension edges) and compute $Score_P(e)$ for each extension edge. To compute $Score_P(e)$ we analyze all reads that map to path P and whose mates map to e. Thus, $Score_P(e)$ reflects our confidence that an extension of the path P by the edge e is correct. We note that to properly map read-pairs and calculate $Score_P(e)$, the total length of path P and edge e should be longer than the insert size.

In addition to function $Score_P(e)$, exSPAnder uses a decision rule $Extend(P)$ that either chooses one of the extension edges to extend the path P or makes the decision to stop growing this path beyond the ending vertex of P. The procedure is iterated over all the paths until no path can be further extended. To initiate this algorithm one can start with a set of single-edge paths formed by all sufficiently long edges in the assembly graph. The resulting paths are output as contigs after removing the paths that are contained within other paths as well as removing non-informative overlaps (i.e. suffixes of paths that represent prefixes of other paths).

This simple approach is merely a framework and, depending on the specifics of the scoring function and the decision rule, it can be either efficient (like in the Ray assembler) or disastrous. The authors of Telescopier made an attempt to improve on Ray’s scoring function and to substantiate it with rigorous statistical analysis. However, scoring functions in both Ray and Telescopier are not universal, e.g. they assume the uniform genome coverage by reads, condition that does not hold for single-cell data.

We demonstrate that exSPAnder works well on single-cell datasets with multiple libraries. We also show that exSPAnder
with (implemented as a part of SPAdes (Bankevich et al., 2012)) improves on existing assemblers on standard bacterial datasets and outperforms such popular assemblers as ABySS (Simpson et al., 2009), Ray (Boisvert et al., 2010), SOAPdenovo (Li et al., 2010) and Velvet (Zerbino and Birney, 2008). ALLPATHS-LG (Gnerre et al., 2011) is an excellent assembler whose applications, however, are limited to specially constructed read-pair libraries. On such libraries EXSPANDER and ALLPATHS-LG generate comparable results (exSPANDER generates longer contigs but ALLPATHS-LG generates longer scaffolds).

2 ANALYSIS OF READ-PAIR LIBRARIES

As we mentioned in the introduction, jumping libraries present additional challenges for genome assembly owing to their high variations in the insert size. Additionally, jumping libraries have high rate of chimeric read-pairs—read-pairs that either have abnormal insert size or incorrect orientation. Chimeric read-pairs further complicate utilization of such libraries while resolving repeats.

Below we present analysis of the insert size distributions for the following data: Brachybacterium faecium isolate dataset (read length 150 bp) and Staphylococcus aureus single-cell dataset (read length 101 bp). Both datasets contain one paired-end and one jumping library. To analyze read-pair libraries we aligned reads to the B. faecium and S. aureus reference genomes using Bowtie 2 (Langmead and Salzberg, 2012) and computed the chimeric read-pairs among all read-pairs. All statistics was obtained using Bowtie 2 (Langmead and Salzberg, 2012). Forward–reverse read-pairs in jumping libraries represent an artifact of the sample preparation and are common for datasets of different types.

Despite the fact that various artifacts of jumping libraries make it difficult to incorporate them into existing assembly tools, EXSPANDER uses jumping libraries to generate high-quality assemblies.

3 EXSPANDER ALGORITHM

EXSPANDER uses an assembly graph constructed by SPAdes (Bankevich et al., 2012) and a set of read-pair libraries. For each library, we map read-pairs to the long edges of the assembly graph and estimate the average insert size along with its confidence interval—a shortest insert size interval that contains at least 95% of all genomic positions.

### Table 1. Information on the B. faecium isolate dataset and the S. aureus single-cell dataset

<table>
<thead>
<tr>
<th>Dataset</th>
<th>B. faecium</th>
<th>S. aureus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library</td>
<td>Paired-end</td>
<td>Jumping</td>
</tr>
<tr>
<td>Number of reads</td>
<td>13 M</td>
<td>41 M</td>
</tr>
<tr>
<td>Average coverage</td>
<td>400x</td>
<td>1100x</td>
</tr>
<tr>
<td>Coverage span</td>
<td>210–570x</td>
<td>0–3000x</td>
</tr>
<tr>
<td>Insert size</td>
<td>270 bp</td>
<td>7.5 kb</td>
</tr>
<tr>
<td>Insert span</td>
<td>150–400 bp</td>
<td>6–10 kb</td>
</tr>
<tr>
<td>Chimeric read-pairs (%)</td>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>Unaligned reads (%)</td>
<td>16</td>
<td>10</td>
</tr>
</tbody>
</table>

Note: Insert span is the shortest insert size interval that contains at least 95% of properly aligned read-pairs. Unaligned reads refer to the percentage of read-pairs that have at least one read unaligned. Chimeric read-pairs refer to the percentage of chimeric read-pairs among all read-pairs. All statistics was obtained using Bowtie 2 (Langmead and Salzberg, 2012). Coverage span is the smallest coverage interval that includes at least 95% of all genomic positions.

#### 3.1 The decision rule

**3.1.1 Single library** Given a path \( P \), we define a *winner* as an edge \( e \) with the maximal score \( \text{Score}_P(e) \) among all extension edges for \( P \). Similarly, a *contender* is defined as an extension edge with the second best score. The winner edge is called the strong winner if (i) \( \text{Score}_P(\text{winner}) > \Theta \) and (ii) \( \text{Score}_P(\text{winner}) > C \cdot \text{Score}_P(\text{contender}) \), where \( \Theta \) and \( C \) are parameters of the algorithm, which are discussed below. If the path \( P \) has a single extension edge (which is obviously the winner), only the first condition is used. The decision rule is defined as follows:

\[
\text{Extend}(P) = \begin{cases} 
  e, & \text{if edge } e \text{ is the strong winner for path } P \\
  \emptyset, & \text{if there is no strong winner for path } P
\end{cases}
\]

**3.1.2 Multiple libraries** The decision rule described above can be generalized for several read-pair libraries. Consider \( M \) read-pair libraries, which are sorted in the order of increasing insert size.
Extend revealed that the smaller is the insert size of a library (and its genome-consecutive edges. We say that an edge \( e \) and \( e' \) form a set of integer points on the 45° line \( y = x - d \) within the rectangle (Fig. 2c). Because the read-pairs from the real sequencing data have variations in the insert size, their corresponding points are typically scattered in the strip between the 45° lines \( y = x - d_{\text{min}} \) and \( y = x - d_{\text{max}} \), where

\[
d_{\text{min}} = \text{Length}(e) - I_{\min} + \text{ReadLength},
\]
\[
d_{\text{max}} = \text{Length}(e) - I_{\max} + \text{ReadLength}.
\]

This strip in the rectangle is further referred to as the confidence strip (Fig. 2d).

Let \( F(x) \) be the empirical distribution of the insert size and \( S \) be a set of all integer points within the confidence strip in the rectangle \((e,e')\). We define the expected number of read-pairs within the confidence strip (under the assumption of the uniform coverage) as

\[
\text{Expected}(e, e') = \sum_{(x,y) \in S} F(x,y),
\]

where \( I(x,y) = \text{Length}(e) - x + y + \text{ReadLength} \) represents the insert size of a read-pair that corresponds to the point \((x,y)\). We also define \( \text{Points}(e,e') \) as the total number of read-pairs (from the real dataset) that correspond to the points within the confidence strip. The notion of density is defined as

\[
\text{Density}(e, e') = \frac{\text{Points}(e,e')}{\text{Expected}(e,e')},
\]

We set \( \text{Density}(e,e') = 0 \), if \( \text{Expected}(e,e') = 0 \). The points outside the confidence strip may represent read-pairs with somewhat larger deviations from the mean insert sizes or chimeric read-pairs. Our analysis revealed that being conservative (e.g. limiting analysis to the confidence strip) allows one to avoid most of the assembly errors caused by chimeric read-pairs, particularly in single-cell projects.

We distinguish between notions of genome-consecutive and graph-consecutive edges and emphasize that graph-consecutive edges are not necessarily genome-consecutive. The decision about which graph-consecutive edges are genome-consecutive is an important part of any assembler. Figure 2d and e illustrate how rectangles help us to make such decisions: both rectangles correspond to graph-consecutive edges, but only rectangle in Figure 2d is formed by the pair of genome-consecutive edges.

The described notions of \( \text{Expected}(e,e') \), \( \text{Points}(e,e') \) and \( \text{Density}(e,e') \) (defined for the case when edges \( e \) and \( e' \) are genome-consecutive) can be generalized for the case when \( e \) and \( e' \) are not genome-consecutive edges under the assumption that genomic distance between them is \( D \). In this case the confidence strip (further referred to as \( \text{Strip}_{\text{p}}(e,e') \)) is bounded by the lines \( y = x - d_{\min} \) and \( y = x - d_{\max} \), where

\[
d_{\min} = D - I_{\min} + \text{ReadLength},
\]
\[
d_{\max} = D - I_{\max} + \text{ReadLength}.
\]

\( \text{Points}_{\text{p}}(e,e') \) similarly represents the number of points within strip \( \text{Strip}_{\text{p}}(e,e') \). Expected\(_{\text{p}}(e,e') \) and Density\(_{\text{p}}(e,e') \) are defined by...
Fig. 3. (a) An example of an assembly graph with the genomic paths \((p_1, p_2, p_3, e)\) and \((p_1, p_2, p_1, e).\) (b, e) The composite rectangles for correct genomic extension of each path: in these cases the points are evenly distributed within the confidence strip and the resulting score is equal to 1. (c, d) The composite rectangles that correspond to incorrect extensions edges of these two paths. In each of these cases, at least one simple rectangle contains few points within the confidence strip using the same formulas and the corresponding confidence strip. Clearly, for two genome-consecutive edges \(Expected(e,e') = Expected_1(e,e')\) and \(Density(e,e') = Density_1(e,e'),\) where \(L = Length(e).\)

The support function reflects whether the number of read-pairs connecting edges \(e\) and \(e'\) supports the conjecture that \(e'\) follows \(e\) in the genome at distance \(D:\)

\[
Support_0(e,e') = \begin{cases} 1, & \text{Density}_0(e,e') > \Psi \\ 0, & \text{otherwise} \end{cases}
\]

where \(\Psi\) is a parameter of the algorithm, which is automatically computed for each read-pair library based on the chimeric read-pair rate (see below). For the standard isolate datasets this parameter corresponds to the coverage cutoff for read-pairs. For single-cell datasets this parameter is usually set to be very low to retain the regions with low coverage, which are typical for single-cell projects. If \(Support_0(e,e') = 1,\) we say that the rectangle \((e,e')\) is supported by the read-pairs.

3.2.2 The naive scoring function To explain the intuition behind \textsc{exspander}, we first introduce the naive scoring function. We further modify the naive scoring function to arrive to the advanced scoring function used in the real \textsc{exspander} implementation.

A path \(P = (p_1, \ldots, p_n)\) and its extension edge \(e\) can be represented as a composite rectangle formed by \(m\) simple rectangles \((p_i, e)\) containing points that correspond to read-pairs connecting edges of \(P\) and \(e.\) Figure 3b shows an example of a composite rectangle, which is formed by a path \((p_1, p_2, p_3)\) and its extension edge \(e_1\) and consists of three simple rectangles. The notion of the confidence strip remains (it now consists of up to \(m\) substrips within simple rectangles), except that it is bounded by the lines \(y = x - d_{\text{min}}\) and \(y = x - d_{\text{max}},\) where

\[
d_{\text{min}} = \text{Length}(P) - I_{\text{min}} + \text{ReadLength},
\]

\[
d_{\text{max}} = \text{Length}(P) - I_{\text{max}} + \text{ReadLength}.
\]

For an edge \(p_j\) from the path \(P\) we define the expected number of points in the confidence substrip within the simple rectangle \((p_j,e)\) as \(Expected_0(p_j,e),\) where \(D_j\) is the distance between start of \(p_j\) and start of \(e\) according to the path \(P\) (i.e. \(D_j = \sum_{i=j}^n Length(p_i)\)). We consider rectangles \((p_j,e), j=\text{start}\ t, \ldots, \text{end}\) such that \(Expected_0(p_j,e) > 0\) and introduce the function \(Score_D(e)\) as the fraction of the total number of expected read-pairs in these rectangles with \(Support_0(p_j,e) = 1:\)

\[
Score_D(e) = \frac{\sum_{j=\text{start}}^{\text{end}} \text{Support}_0(p_j,e) \cdot Expected_0(p_j,e)}{\sum_{j=\text{start}}^{\text{end}} Expected_0(p_j,e)}.
\]

We set \(Score_D(e) = 0\) if all simple rectangle have zero expected read-pairs.

Figure 3a shows paths \(P = (p_1, p_2, p_3)\) and \(P' = (p_1, p_2, p_3)\) and its extension edges \(e_1, e_2.\) Let \((p_1, p_2, p_3, e_1)\) and \((p_1, p_2, p_3, e_2)\) be the true (but unknown) genomic paths. Figure 3b shows the composite rectangle for path \(P\) and its correct extension \(e_1,\) in which points within the confidence strip are rather evenly distributed resulting in \(Score_D(e_1) = 1.\) Figure 3c shows the composite rectangle for path \(P\) and its incorrect extension edge \(e_2.\) Because \((p_1, p_2, p_3, e_2)\) is a genomic path, density of the points in the sectors of the confidence strip corresponding to edges \(p_1\) and \(p_3\) is high. However, edge \(p_2\) of the path \(P'\) does not support extension edge \(e_2\) because there are few points in the rectangle \((p_2, e_2).\) Additionally, Figure 3d and e shows composite rectangles for all possible extension edges for path \(P' = (p_1, p_2, p_3)\).

Because the defined scoring function does not linearly depend on read coverage, it is well suitable for both single-cell and standard sequencing projects. At the same time, considering only read-pairs with insert size in \([I_{\text{min}}, I_{\text{max}}]\) (which correspond to points within the confidence strip) allows one to filter out most of the chimeric read-pairs (common for single-cell datasets) and to minimize their influence on the scoring function.

3.2.3 The advanced scoring function The naive scoring function \(Score_D(e)\) described above works well in many cases but may be too conservative when the path \(P\) contains repetitive edges (edges that are visited more than once by the genomic traversal). Figure 4 illustrates the case when the path \(P\) has a repetitive edge and motivates the need for further improvements in the scoring function.

Figure 4a shows an assembly graph with four unique edges \((p, p', e\) and \(e')\) and a single repetitive edge \(p_i\) with multiplicity 2. We assume that paths \((p, p, e)\) and \((p', p', e)\) are genomic and paths \((p, p, e)\) and \((p', p', e)\) are non-genomic. Our goal is to design an algorithm that correctly extends genomic paths \((p, p)\)
and \((p',p_e)\) into longer genomic paths \((p,p,e)\) and \((p',p,e')\), respectively.

Consider a path \(P = (p,p_e)\) and composite rectangles \((P,e)\) and \((P,e')\) (Fig. 4b and c). As Figure 4 illustrates, Score\(_{C}(e)\) is similar to Score\(_{C}(e')\), implying that there is no strong winner for path \(P\) and preventing us from extending the path \(P\) by edge \(e\). However, because the repetitive edge \(p\) supports both extension edges \(e\) and \(e'\), it does not provide any valuable information about the correct extension of the path \(P\). Therefore, to make a decision about extending the path \(P\) by an extension edge \(e\), we should have excluded \(p\) from the consideration as a repetitive edge. Because we do not know in advance which edges of the assembly graph correspond to repeats in the genome, we classify edge \(p\) as repetitive because it supports both extension edges \(e\) and \(e'\).

Below we present the ExSPAnDer algorithm that allows us to exclude repetitive edges from contributing to scoring. An extension edge \(e\) of path \(P\) is called an active edge if \(C\cdot\text{Score}\_C(e) > \text{Score}\_C(\text{winner})\). At the first step of the algorithm we score all extension edges and form a set of active edges \(A\). An edge \(p\) in path \(P\) is classified as repetitive if it supports all active edges, i.e., \(\text{Support}_{D}(p) = 1\) for all \(e\) in \(A\). At the second step we mark all repetitive edges \(p\) in \(P\) and recalculate scores of all edges in \(A\) ignoring these repetitive edges. We then update \(A\) by removing all non-active edges and iterate the process. The process continues until yet another iteration does not change the set of active edges \(A\). If \(A = \{e\}\) and \(\text{Score}\_C(e) > \Theta\) (which means that \(e\) satisfies both conditions in the decision rule) the extension edge \(e\) is considered to be a strong winner and added to the path \(P\). Otherwise, we stop extending path \(P\).

We further demonstrate the work of the ExSPAnDer algorithm using a simple assembly graph shown in Figure 5. The paths \((p_1, p_2, p_3, e)\), \((p'_1, p_2, p_3, e)\) and \((p'_2, p_3, e)\) are genomic paths, which means that edges \(p_2\) and \(p_3\) are repetitive and have multiplicities 2 and 3, respectively.

Let \(P = (p_1, p_2, p_3)\) be a path we aim to extend. We first calculate scores of all extension edges using the composite rectangles (Fig. 6a-c) and form a set of active edges \(A = \{e_1, e_2, e_3\}\) based on their scores (marked red in Fig. 6d). Because \(\text{Support}_{D}(p) = 1\) for \(i = 1, 2, 3\), edge \(p_3\) is classified as repetitive and is removed from further consideration (Fig. 6e). We now recalculate scores for the extension edges in \(A\) ignoring repetitive edge \(p_3\) (Fig. 6f–h) and remove non-active edge \(e_3\) from \(A\) (Fig. 6i). Using the updated set \(A = \{e_1, e_2\}\) we again proceed to the repeat detection step and mark edge \(p_2\) as repetitive because \(\text{Support}_{D}(p_2, e_i) = 1\) for \(i = 1, 2\) (Fig. 6j). Finally, we once again recalculate scores of the extension edges in \(A\) (Fig. 6k–m) and remove \(e_2\) as non-active (Fig. 6n). The extension edge \(e_1\) remains the only active edge in and is used to extend path \(P\).

Extensive tests of the advanced scoring function revealed that it works well across diverse datasets including single-cell jumping libraries with high variations in the insert size, extremely non-uniform coverage and large number of chimeric reads and chimeric read-pairs (see Section 4).

### 3.3 Scaffolding

After all paths are constructed, we consider all pairs of paths that form composite rectangles with non-zero number of points (Fig. 7). For each such pair of paths \(P\) and \(P'\) we can check whether points in the corresponding composite rectangle are scattered around a certain 45° line using SPAdes distance estimation procedure (Nurk et al., 2013). When SPAdes provides the estimated distance \(D\) between \(P\) and \(P'\), we use ExSPAnDer to verify the conjecture that \(P'\) follows \(P\) at distance \(D\). If this conjecture is supported and does not contradict to any other conjectures about these paths, we extend the path \(P\) by \(P'\) (the scaffolding step). We estimate the gap length between the paths as \(D - \text{Length}(P)\) and insert the appropriate number of ‘N’ symbols (unspecified nucleotide) between end of \(P\) and start of \(P'\). If paths \(P\) and \(P'\) overlap, we construct their overlap alignment to correct distance \(D\).

### 3.4 Choice of the parameters

#### 3.4.1 The scoring function

We select the parameter \(\Psi\) as a threshold for the density of the read-pairs within the confidence strip. We therefore assume that rectangles with the density below \(\Psi\) contain mostly chimeric read-pairs and should be ignored while calculating the score of an extension edge. To select \(\Psi\) for a particular read-pair library, we estimate the distribution of the densities for rectangles that contain only chimeric read-pairs (false rectangles) and for rectangles that contain only non-chimeric read-pairs (true rectangles).

To partition all read-pairs into chimeric and non-chimeric, one needs the complete genome that is unavailable. To get around this, we identify a subset of chimeric reads using the long edges in the assembly graph (e.g. edges longer than NS0) that can be
Fig. 6. A step-by-step example of the exSPANDER algorithm. (a-c) Forming a set of active edges \{e_1, e_2, e_3\} (marked red) for the path \(P = (p_1, p_2, p_3)\) using the corresponding composite rectangles. (d, e) Classifying of edge \(p_3\) as repetitive and removing it from further consideration (marking gray). Edges that are not classified as repetitive are colored in blue. (f-h) Recalculating scores of the extension edges and updating the set of active edges. (i, j) Removing repetitive edge \(p_2\). (k-m) Recalculating scores for the remaining active edges \{e_1, e_2\} and removing \(e_2\) as non-active. (n) Selecting the only active edge \(e_1\) as an extension for the path \(P\).
viewed as subgenomes of the complete genome. By mapping a read-pair to a long edge we can compute its insert size and thus classify read-pairs with abnormally large insert sizes as chimeric.

To estimate the parameter $\Psi$, we generate a large number of equally sized rectangles by artificially ‘splitting’ each long edge into shorter edges of equal length (e.g. 100 bp) called uni-edges. We then consider rectangles formed by all pairs of uni-edges within the same long edge. Because we know the exact distance between such uni-edges, we can calculate the expected number of read-pairs $ExpectedD(e_1, e_2)$ for all pairs of uni-edges within the same long edge. If uni-edges $e_1$ and $e_2$ come from the same long edge and $ExpectedD(e_1, e_2) = 0$, then all points in the confidence strip $StripD(e_1, e_2)$ represent chimeric read-pairs connecting $e_1$ and $e_2$. Such edge-pair $(e_1, e_2)$ is classified as a false edge-pair. Otherwise, if $ExpectedD(e_1, e_2) > 0$, we assume that all points within $StripD(e_1, e_2)$ represent correct read-pairs and classify $(e_1, e_2)$ as a true edge-pair.

Edges $e_1$ and $e_2$ and a parameter $D$ define the strip $StripD(e_1, e_2)$ with the number of points $PointsD(e_1, e_2)$. For a pair of edges $e_1$ and $e_2$, we define

$$D^* = \arg\max_D PointsD(e_1, e_2).$$

When $(e_1, e_2)$ is a false edge-pair, $D^*$ defines a confidence strip $StripD^*(e_1, e_2)$ with the maximum number of chimeric read-pairs. To compute the threshold $\Psi$, we assume this worst-case scenario for all pairs of uni-edges (within the same long edge) by using distance $D^*$ (rather than the known genomic distance) for calculating the densities $DensityD^*(e_1, e_2)$.

For a certain value of $\Psi$ we define false positives (false negatives) as the false (true) edge-pairs that have density higher (lower) than $\Psi$. Figures 8a and b illustrate how false-positive rate (green) and false-negative rate (blue) depend on the parameter $\Psi$. ExSPAnder selects $\Psi$ that corresponds to the intersection point of the false-positive and false-negative plots. Our benchmarking revealed that such choice of the parameter $\Psi$ allows one to filter out the rectangles containing only chimeric reads-pairs based on their densities. Additional analysis revealed that estimating the parameter $\Psi$ is an important step in the ExSPAnder algorithm because varying this parameter may significantly affect the assembly quality (see Section 3 in Appendix).

3.4.2 The decision rule Our analysis revealed that varying parameters $C$ and $\Theta$ within specified ranges (see Section 3 of the Appendix) hardly affects the quality of the resulting assemblies. However, selecting inappropriate $C$ and $\Theta$ may result in a deteriorated performance of ExSPAnder. Thus, we arbitrarily select these parameters within the ranges specified in Section 3 of Appendix (analysis of diverse sequencing datasets supports the default values $C = 1.5$ and $\Theta = 0.5$).

4 RESULTS

4.1.1 Datasets We have compared ExSPAnder (coupled with SPAdes assembler) with several popular assemblers on the $B$.faecium isolate dataset (genome size 3.6 Mb) and the $S$.aureus single-cell dataset (genome size 2.9 Mb). For each dataset we have generated assemblies of (i) only paired-end library and (ii) both paired-end and jumping libraries. Section 1 of Appendix provides a detailed description of both datasets.

4.1.2 Benchmarking assemblies ABySS 1.3.6, Ray 2.0.0, Velvet, Velvet-SC and SOAPdenovo 2.0.4 were run with $k$-mer size 55. IDBA-UD 1.1.1 was run in its default iterative mode. The authors of (Peng et al., 2012) released this new version of IDBA-UD that is capable of using several paired-end libraries, but there is no manuscript yet covering this new development. ALLPATHS-LG was run with the default parameters; however, we down-sampled jumping library for the $B$.faecium dataset to generate 100× coverage required by ALLPATHS-LG. SPAdes 2.4 (previous version of SPAdes that did not include ExSPAnder and did not support multiple libraries) was run in its default iterative mode with $k = 21, 33, 55, 77$ for the $B$.faecium dataset and $k = 21, 33, 55$ for the $S$.aureus dataset. ExSPAnder (coupled with SPAdes) was run using the default parameters.

To analyze the resulting assemblies we used QUAST 2.2 (Gurevich et al., 2013) that reports various parameters including NG50 (similar to N50, but is calculated with respect to the reference genome size), the total number of contigs/scaffolds, the length of the longest assembled contig/scaffold, the number of misassemblies and the fraction of genome mapped. QUAST defines a misassembly breakpoint as a position in the contig/scaffold, such that its left and right flanking sequences either align to the reference genome over 1 kb away from each other, or overlap by >1 kb, or align on opposite strands or different chromosomes (Gurevich et al., 2013). To compare assemblers we used both contigs and scaffolds of length exceeding 500 bp.

![Fig. 7. An example of a composite rectangle formed by paths $p_1, p_2, p_3$ and $p_1', p_2', p_3'$](image1)

![Fig. 8. Plots of the false-positive (green) and false-negative (blue) rates for (a) $B$.faecium and (b) $S$.aureus paired-end libraries](image2)
Tables 2 and 3 show the benchmarking results for the \textit{B.\textit{faecium}} isolate dataset. Interestingly, the single-cell assemblers (IDBA-UD and \textsc{exSPander} coupled with \textsc{SPAdes}) as well as \textsc{ABySS} performed well on the \textit{B.\textit{faecium}} isolate dataset and produced contigs with the largest NG50 in the case of a single library. While \textsc{ABySS} generated the assembly with the maximal number of misassemblies (\textsc{Velvet} and \textsc{Velvet-SC} are not included in the benchmark experiment for single-cell data), Velvet and \textsc{Velvet-SC} are not included in the benchmark experiment for single-cell data. Velvet and \textsc{Velvet-SC} are not included in the benchmark experiment for single-cell data.

In the case of two libraries, \textsc{exSPander} produced the best contigs while \textsc{ALLPATHS-LG} produced the best scaffolds. The complexity of using jumping libraries is reflected in a deteriorated performance of \textsc{ABySS} and \textsc{Ray} (reduction in NG50) as well as \textsc{Velvet} and \textsc{Velvet-SC} (dramatic increase in the number of misassemblies).

Tables 4 and 5 compare various assemblers on the \textit{S.\textit{aureus}} single-cell dataset. This comparison highlights the complexity of both (i) assembling single-cell datasets and (ii) using jumping libraries. For example, \textsc{SOAPdenovo} produced assemblies of poor quality for single-cell data (we decided not to include it in Tables 4 and 5). Similarly, \textsc{ABySS} produced assemblies with high number of misassemblies for the single-cell data. \textsc{Velvet} and \textsc{Velvet-SC} are not included in the benchmark experiment for jumping libraries testing because they also produce low-quality assemblies when both paired-end and jumping libraries are used simultaneously. \textsc{IDBA-UD} performed well on a single paired-end library, but produced an assembly of lower quality when both libraries were provided (decreased NG50). \textsc{exSPander} produced assemblies with the highest NG50 and largest assembled contig/scaffold.

Using only paired-end library \textsc{IDBA-UD}, \textsc{SPAdes} 2.4 and \textsc{exSPander} recovered the largest fraction of the genome (>98.5%). However, the highest genome fraction of the assembly produced assemblies with the highest NG50 and largest assembled contig/scaffold.
generated by SPAdes 2.4 reflects the specifics of SPAdes 2.4 and QUAST reporting (some artifacts with reporting of repetitive regions) rather than real advantage of SPAdes 2.4 with respect to this parameter.

When using both libraries simultaneously, exSPANDER produced assemblies with the highest genome fraction exceeding 99%, the largest genome fraction we saw across dozens of single-cell datasets assembled with SPAdes in the past 2 years. Moreover, Tables 4 and 5 show that exSPANDER successfully deals with the high rate of the chimeric read-pairs and relatively high variations in the insert size.

5 CONCLUSION

We have presented exSPANDER algorithm for resolving repeats using either a single or multiple read-pair libraries with different insert sizes, which is applicable for both single-cell and isolate bacterial datasets. Benchmarks across eight popular assemblers demonstrate that exSPANDER produces high-quality assemblies for datasets of different types. Additionally, as illustrated by recent integration of Illumina and PacBio reads in SPAdes 3.0, exSPANDER is a flexible approach that can be easily modified to work with diverse types of sequencing data.

Table 5. Comparison of scaffolds for the S.aureus single-cell dataset

<table>
<thead>
<tr>
<th>Assembler</th>
<th>NG50</th>
<th>Number of scaffolds</th>
<th>Largest</th>
<th>Number of mis</th>
<th>GF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Only paired-end library</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ABySS</td>
<td>28</td>
<td>910</td>
<td>91</td>
<td>270</td>
<td>98.2</td>
</tr>
<tr>
<td>Ray</td>
<td>21</td>
<td>306</td>
<td>108</td>
<td>14</td>
<td>88.7</td>
</tr>
<tr>
<td>Velvet-SC</td>
<td>10</td>
<td>538</td>
<td>56</td>
<td>2</td>
<td>93.2</td>
</tr>
<tr>
<td>IDBA-UD</td>
<td>88</td>
<td>382</td>
<td>161</td>
<td>8</td>
<td>98.2</td>
</tr>
<tr>
<td>SPAdes 2.4</td>
<td>99</td>
<td>391</td>
<td>230</td>
<td>8</td>
<td>99.2</td>
</tr>
<tr>
<td>exSPANDER</td>
<td>148</td>
<td>357</td>
<td>426</td>
<td>4</td>
<td>98.6</td>
</tr>
<tr>
<td>Both paired-end and jumping libraries</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ABySS</td>
<td>30</td>
<td>852</td>
<td>91</td>
<td>275</td>
<td>98.0</td>
</tr>
<tr>
<td>ALLPATHS-LG</td>
<td>40</td>
<td>165</td>
<td>132</td>
<td>69</td>
<td>79.9</td>
</tr>
<tr>
<td>Ray</td>
<td>100</td>
<td>169</td>
<td>486</td>
<td>25</td>
<td>93.5</td>
</tr>
<tr>
<td>IDBA-UD</td>
<td>55</td>
<td>397</td>
<td>161</td>
<td>9</td>
<td>98.6</td>
</tr>
<tr>
<td>exSPANDER</td>
<td>314</td>
<td>302</td>
<td>603</td>
<td>9</td>
<td>99.3</td>
</tr>
</tbody>
</table>

Note: NG50 is given in kb; number of scaffolds is the total number of scaffolds >500 bp; largest stands for the length (in kb) of the longest scaffold assembled; number of mis is the number of misassemblies; GF stands for the fraction of genome mapped given in percent. In each column, the best value is indicated in bold.

ACKNOWLEDGEMENTS

The B.faecium sequencing data were produced by the Joint Genome Institute in collaboration with the user community. The S.aureus dataset was provided by the Human Microbiome Project (bioproject ID PRJNA236734).

Funding: This work was supported by the Government of the Russian Federation [grant numbers 11.G34.31.0018, 11.G34.31.0068]; the National Institutes of Health [3P41RR024851-02S1]; and the Russian Fund for Basic Research [12-01-00747-a to A.K.].

Conflict of Interest: none declared.

REFERENCES