Bioinformatics, 31(20), 2015, 3262–3268
doi: 10.1093/bioinformatics/btv337
Advance Access Publication Date: 3 June 2015
Original Paper

Sequence analysis

Assembling short reads from jumping libraries with large insert sizes

Irina Vasilinetc[1], Andrey D. Prjibelski[1,2,*], Alexey Gurevich[1,2], Anton Korobeynikov[1,2,3] and Pavel A. Pevzner[2,4]

[1]Algorithmic Biology Lab, St. Petersburg Academic University 194021, [2]Center for Algorithmic Biotechnology, Institute of Translational Biomedicine, St.Petersburg State University, 199004, [3]Department of Mathematics and Mechanics, St. Petersburg State University, St. Petersburg, 198504, Russia and [4]Department of Computer Science and Engineering, University of California, San Diego, CA 92093-0404, USA

*To whom correspondence should be addressed.

Associate Editor: Inanc Birol

Received on February 12, 2015; revised on May 3, 2015; accepted on May 26, 2015

Abstract

Motivation: Advances in Next-Generation Sequencing technologies and sample preparation recently enabled generation of high-quality jumping libraries that have a potential to significantly improve short read assemblies. However, assembly algorithms have to catch up with experimental innovations to benefit from them and to produce high-quality assemblies.

Results: We present a new algorithm that extends recently described exSPAnder universal repeat resolution approach to enable its applications to several challenging data types, including jumping libraries generated by the recently developed Illumina Nextera Mate Pair protocol. We demonstrate that, with these improvements, bacterial genomes often can be assembled in a few contigs using only a single Nextera Mate Pair library of short reads.

Availability and implementation: Described algorithms are implemented in C++ as a part of SPAdes genome assembler, which is freely available at bioinf.spbau.ru/en/spades.

Contact: ap@bioinf.spbau.ru

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

1 Introduction

In an article titled 'De novo fragment assembly with short mate-paired reads: Does the read length matter?', Chaisson *et al.* (2009) argued that availability of paired reads with *long* and *accurate* insert sizes (rather than the increase in the read length) is the most important factor for improving the quality of short read assemblies. However, while paired reads with long insert sizes have been extensively used in many assembly projects, robust generation of read-pairs with accurate insert sizes proved to be difficult and have only been achieved recently.

The recently emerged sample preparation technologies open new opportunities for genome assembly from short reads. For example, Illumina Nextera Mate Pair protocol generates long inserts (3 kb and longer) that feature rather tight insert size distribution and small rate of non-circularized fragments that result in read-pairs with abnormal distances. As discussed in Chaisson *et al.* (2009), such read-pair libraries may enable assemblies approaching the quality of assemblies from long reads of length equal to the insert size. Moreover, they can potentially substitute the existing assembly approaches based on a combination of *short* paired-end libraries (with insert size less than 1 kb) and *long* jumping libraries (with insert sizes typically longer than 1 kb) by a pipeline based on a single Nextera Mate Pair library.

However, even though the popular assembly algorithms perform well with the previously proposed approaches to sample preparation, they have not kept up with recent experimental innovations. To catch up, bioinfomaticians either need to design novel tools for every technology improvement or to develop a *universal* assembler that can be easily modified to support new data types. For example, Ray (Boisvert *et al.*, 2010) and SPAdes (Bankevich *et al.*, 2012)

[with the recently implemented exSPAnder (Prjibelski *et al.*, 2014) algorithm] use similar approach to repeat resolution that can be adapted for new types of sequencing data. However, the recently proposed Nextera Mate Pair libraries present new computational challenges that go beyond the algorithmic framework of both Ray and SPAdes. In this work, we describe several algorithmic advances to exSPAnder that allows one to efficiently utilize jumping libraries and to perform assembly using only high-quality Nextera Mate Pair libraries.

The exSPAnder algorithm (Prjibelski *et al.*, 2014) is based on the *path extension* framework that was proposed by the authors of the Ray assembler (Boisvert *et al.*, 2010) and later implemented in the Telescoper (Bresler *et al.*, 2012) and PERGA (Zhu *et al.*, 2014) assemblers. Given a path P in the assembly graph (Bankevich *et al.*, 2012; Prjibelski *et al.*, 2014), exSPAnder iteratively attempts to grow it by choosing one of its *extension edges* (all edges starting at the terminal vertex of the path P). The assembly graph is defined as simplified de Bruijn graph (Pevzner *et al.*, 2001) of k-mers in reads after removal of *bulges*, *tips* and *chimeric edges*.

To extend a path P, exSPAnder computes the scoring function $\text{Score}_P(e)$ for each extension edge e using read-pairs with one read mapping to P and another read mapping to e (further referred to as (P, e)-*connecting read-pairs*). Afterward, exSPAnder decides whether to select the top-scoring extension edge or to stop growing P. It iteratively repeats the path extension procedure starting with single-edge paths until every edge in the assembly graph is covered by at least one path and no path can be extended further. To generate equivalent contigs on both strands, exSPAnder is implemented as a bidirectional approach that can extend a path in both directions.

While the scoring function $\text{Score}_P(e)$ described in Prjibelski *et al.* (2014) works well with short libraries, it appears to be rather inefficient when using jumping libraries. The key limitation of the previously defined scoring function is that it analyses only (P, e)-connecting read-pairs (where e is an extension edge of path P) and ignores read-pairs that connect path P with other edges. When an edge e is short and the variations in the insert sizes are large, there is a danger that no (P, e)-connecting read-pairs exist and thus $\text{Score}(P, e) = 0$ even if e is the correct extension edge (Fig. 1). Thus, the decision rule may stop extending path P or even select an incorrect extension edge. Additionally, the approach described in Prjibelski *et al.* (2014) is inapplicable for scaffolding procedure, since it is unable to jump over coverage gaps. In this article, we extend the exSPAnder approach to scaffolding. This extension is important since scaffolding with jumping libraries may dramatically improve the assembly quality.

We describe several algorithms that address these bottlenecks based on the following idea. Consider a set of *extension paths* $\mathcal{E}$ [rather than extension edges as in Prjibelski *et al.* (2014)] that contain all sufficiently long paths (longer than the insert size) starting from the extension edges of the path P. Once the set $\mathcal{E}$ is constructed, we choose the best-scoring path E in $\mathcal{E}$ and extend path P by the first edge of E. Our analysis has shown that such conservative extension (by the first edge of the best-scoring extension path rather than by the entire path) provides more accurate assemblies. To perform scaffolding procedure, we allow extension paths to 'jump' over coverage gaps in the assembly graph.

This intuitive approach, while appealing, is often impractical since the assembly graph is usually tangled, resulting in a prohibitively large number of extension paths. To reduce the running time, we have implemented the new algorithm based on the observation that, instead of the exhaustive search through the set of all extension paths, one can significantly prune this set using single reads and paired-end libraries (if available).

We demonstrate that the new algorithm enables assemblies of nearly complete genomes from a *single* Nextera Mate Pair library. We also show that SPAdes, coupled with the improved exSPAnder algorithm, outperforms other popular assemblers, such as ABySS (Simpson *et al.*, 2009), IDBA-UD (Peng *et al.*, 2012), Ray (Boisvert *et al.*, 2010), SOAPdenovo (Li *et al.*, 2010) and Velvet (Zerbino and Birney, 2008) on various types of datasets.

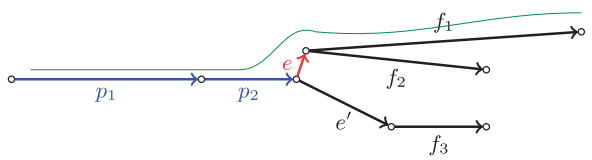

Fig. 1. An example of the assembly graph with path $P = (p_1, p_2)$ (blue) and its correct short extension edge e (red). In this case, there may be no (P, e)-connecting read-pairs, but there may be (P, f_1)-connecting read-pairs since the correct genomic path includes the path (p_1, p_2, e, f_1), which is shown in green above the graph

2 Methods

2.1 The exSPAnder framework

2.1.1 Rectangles

exSPAnder utilizes the concept of the *rectangle graph* introduced by Bankevich *et al.* (2012) and further developed in (Vyahhi *et al.*, 2012). Let e and e' be edges in the *assembly graph* (see (Bankevich *et al.*, 2012) for the construction of the assembly graph) separated by a known distance D and (r, r') be a read-pair, such that read r maps to e at position x_0 and read r' maps to e' at position y_0 (Fig. 2a). We note that while D is not known a priori, exSPAnder only considers edges that are connected by a previously constructed path, which unambiguously defines distance D.

Figure 2b shows the read-pair (r, r') represented as a point (x_0, y_0) within the rectangle formed by the edges e and e'. We further refer to this rectangle simply as (e, e'). Because of variations in the insert sizes, real read-pairs connecting edges e and e' typically correspond to the points that are scattered in the *confidence strip*—a strip between the 45° lines $y = x - d_{\min}$ and $y = x - d_{\max}$ within the rectangle (e, e') (Fig. 2c), where

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

Here, $[I_{\min}, I_{\max}]$ is the *confidence interval* of the insert size distribution defined as the smallest insert size interval that contains at least 80% of read-pairs. Since reads may have variable lengths (e.g. after quality trimming or error correction), we set ReadLength as the maximal read length across all reads in the current library.

2.1.2 The decision rule

The decision rule relies on the scoring function that will be described in the next section. To extend a path P, we score all extension edges $e_1, \ldots, e_n$ and select e_i as a correct extension if the following conditions are met:

1. $\text{Score}_P(e_i) > C \cdot \text{Score}_P(e_j)$ for all $j \neq i$
2. $\text{Score}_P(e_i) > \Theta$

Here, C and Θ are the exSPAnder parameters described in (Prjibelski *et al.*, 2014). If no extension edge meets these conditions, the algorithm stops growing path P.

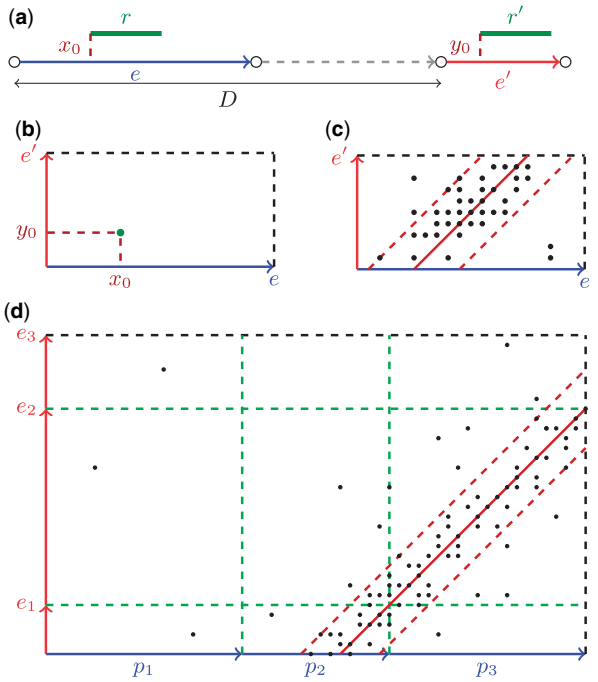

Fig. 2. (**a**) A read-pair (r, r') mapping to edges e and e' in the assembly graph at positions x_0 and y_0, respectively. (**b**) Read-pair (r, r') is represented as a point (x_0, y_0) in the rectangle (e, e'). (**c**) Since the insert size varies, most read-pairs correspond to the points scattered within the confidence strip in the rectangle (e, e'). (**d**) An example of a composite rectangle formed by paths $P = (p_1, p_2, p_3)$ and $E = (e_1, e_2, e_3)$, which consists of nine simple rectangles. Note that the confidence strip crosses only five out of nine simple rectangles. The points outside the confidence strip appear in eight out of nine simple rectangles and represent read-pairs with abnormal insert sizes

In the case of multiple read-pair libraries, we process them in the order of increasing insert sizes until a certain library provides an extension edge satisfying the decision rule. We stop growing path P if no library provides an extension edge satisfying this rule. Processing libraries in a step-by-step fashion (rather than incorporating all libraries at once) has proven to result in more accurate and continuous assemblies.

2.2 Scoring function

Given a path $P = (p_1, p_2, \ldots, p_n)$ and its extension path $E = (e_1, e_2, \ldots, e_m)$, the composite rectangle for paths P and E is formed by $n \cdot m$ simple rectangles (p_i, e_j) (Fig. 2d). The distance D_{ij} between a pair of edges p_i and e_j according to the paths P and E is equal to

$$D_{ij} = \sum_{k=i}^{n} \text{Length}(p_k) + \sum_{l=1}^{j-1} \text{Length}(e_l).$$

We define the following variables [see Prjibelski *et al.* (2014) for more details]:

- $\text{Expected}_{D_{ij}}(p_i, e_j)$: the expected number of (p_i, e_j)-connecting read-pairs;
- $\text{Observed}_{D_{ij}}(p_i, e_j)$: the observed number of (p_i, e_j)-connecting read-pairs.

To filter out rectangles filled by chimeric read-pairs, we use the *support function*

$$\text{Support}_{D_{ij}}(p_i, e_j) = \begin{cases} 1, & \text{Density}_{D_{ij}}(p_i, e_j) > \Psi \\ 0, & \text{otherwise} \end{cases}$$

where the density value is defined as

$$\text{Density}_{D_{ij}}(p_i, e_j) = \frac{\text{Observed}_{D_{ij}}(p_i, e_j)}{\text{Expected}_{D_{ij}}(p_i, e_j)}.$$

Thus, exSPAnder ignores rectangles with a density lower than a pre-defined threshold Ψ.

While this approach worked well for short libraries, our benchmarking revealed that it deteriorates for long jumping libraries that typically have a higher rate of read-pairs with abnormal insert sizes (referred to as *chimeric read-pairs*).

We thus define a new support function that evaluates whether a simple rectangle (p_i, e_j) within the composite rectangle (P, E) can be considered as trusted and used for calculating the scoring function:

$$\text{Support}_{D_{ij}}(p_i, e_j) = \begin{cases} 1, & \text{Observed}_{D_{ij}}(p_i, e_j) > \eta \\ 0, & \text{otherwise} \end{cases}$$

where η is a user-controlled parameter with very conservative default value $\eta = 30$.

While it is not clear how to select optimal η, our analysis of the distribution of the number of chimeric points within all rectangles across a wide range of bacterial genomes revealed that hardly any rectangles contain more than 30 chimeric points. We note that by setting the parameter η, one essentially ignores the contribution from small rectangles containing less than η points. While it may appear that we unfairly ignore small rectangles, our benchmarking revealed that this approach actually improves the assembly quality. Moreover, the users can change the parameter η in the case of assembly projects with unusually low or high coverage.

When the extension path contains a single edge e, exSPAnder uses the following path-edge scoring function:

$$\text{Score}_P(e) = \frac{\sum_{i=1}^{n} \text{Support}_{D_i}(p_i, e) \cdot \text{Expected}_{D_i}(p_i, e)}{\sum_{i=1}^{n} \text{Expected}_{D_i}(p_i, e)}.$$

In this article, we generalize this path-edge scoring function to a path–path scoring function as follows:

$$\text{Score}_P(E) = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m} \text{Support}_{D_{ij}}(p_i, e_j) \cdot \text{Expected}_{D_{ij}}(p_i, e_j)}{\sum_{i=1}^{n}\sum_{j=1}^{m} \text{Expected}_{D_{ij}}(p_i, e_j)}.$$

Given a confidence interval $[I_{\min}, I_{\max}]$ of the insert size distribution, we call a path *long* if its length exceeds $I_{\max}$ and *short* otherwise. The algorithm that we describe below limits attention to the long extension paths $(e_1, e_2, \ldots, e_m)$ whose prefix $(e_1, e_2, \ldots, e_{m-1})$ is short. Given a path P, for each extension edge e_i of P, we construct a set of extension paths $\mathcal{E}$. A top-scored path E in $\mathcal{E}$ is called a *representative* path for edge e (Fig. 3a). The score of the extension edge e is now defined as the score of its representative path, i.e. $\text{Score}_P(e) = \text{Score}_P(E)$.

The new approach to define $\text{Score}_P(e)$ can be applied to the libraries with both short and long insert sizes. While the resulting improvements are significant for long insert sizes, they appear to be minor for the libraries with small insert sizes. Thus, in the default exSPAnder mode, the new scoring function is used only for the jumping libraries.

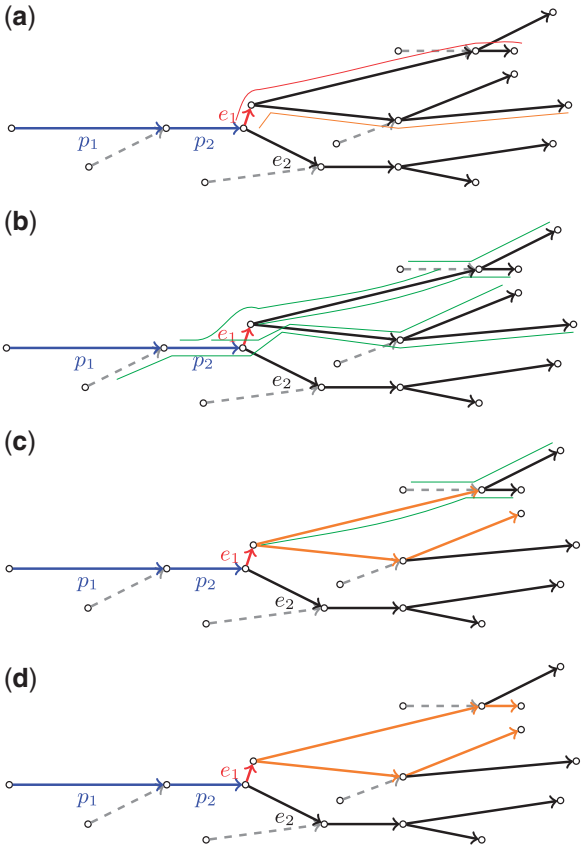

Fig. 3. (**a**) An example of path P in the assembly graph (shown in blue) and its extension edges e_1 and e_2. The representative path for e_1 is shown in red above the graph, whereas another extension path for e_1 (with lower score) is shown in orange. (**b–d**) A step-by-step example of the work of *ExtensionSearch* algorithm for constructing extension paths (marked orange) for edge e_1 using the pre-constructed paths from the set $\mathcal{S}$ (shown as five green paths in the top figure). At each step, the *ExtensionSearch* algorithm grows an extension path E (that is shorter than I_{max}) using only E-consistent extension edges (with respect to $\mathcal{S}$)

2.3 Constructing extension paths

In the case of long libraries, the Depth-First Search (DFS) algorithm for exploring all paths and selecting representative paths becomes rather slow, e.g. the number of extension paths can reach millions even for a bacterial genome. To decrease the number of extension paths, we developed the *ExtensionSearch* algorithm that uses a set of *pre-constructed paths* $\mathcal{S}$ that are generated by the previous exSPAnder algorithm (Prjibelski *et al.*, 2014) using single reads and short paired-end libraries if they are available.

We use the notation Suffix(E, l) [and Prefix(E, l)] to refer to a path formed by the last (first) l edges of a path E. For a path E and a set of pre-constructed paths $\mathcal{S}$, we say that an extension edge e of path E is *E-consistent* (with respect to $\mathcal{S}$) if $\mathcal{S}$ contains a path E', such that $\text{Suffix}(E+e, l) = \text{Prefix}(E', l)$ for some $l > 0$. Instead of extending a path E by all its extension edges (like in the DFS algorithm), *ExtensionSearch* uses the following heuristics to construct extension paths:

- Extend a path E only by E-consistent extension edges;
- Extend only paths that are shorter than I_{max}.

As the result, we obtain a set of extension paths that 'comply' with the set of pre-constructed paths $\mathcal{S}$ with length at least I_{max} (Fig. 3b–d).

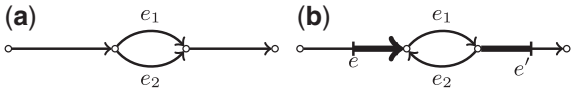

Fig. 4. An example of a simple bulge (**a**) and simple loop (**b**) consisting of edges e_1 and e_2. A simple bulge (loop) is formed by two parallel (anti-parallel) edges between two vertices. Regions used for calculating the local coverage in the vicinity of the loop are marked with bold segments

ExtensionSearch greatly reduces the number of the extension paths when compared with the DFS algorithm. However, in the tangled regions of the assembly graph with many bulges (corresponding to variations in repeats) and loops (corresponding to tandem duplications), the number of paths is still high (Fig. 4). For example, each bulge potentially doubles the number of extension paths explored by the exSPAnder algorithm (e.g. *ExtensionSearch* can select either of edges e_1 and e_2 in Fig. 4a). Each loop potentially multiples the number of extension paths by the (unknown) *multiplicity of the loop* (i.e. the number of times the loop traverses the edge e_2 in Fig. 4b). In the *ExtensionSearch*, we focus only on simple bulges (Fig. 4a) and simple loops (Fig. 4b) since they represent the majority of all cases that may trigger the increase in the running time of the algorithm.

To limit the explosion of the extension paths, we add the following constrains to the *ExtensionSearch* algorithm:

- For each bulge formed by two edges, only the top-scoring edge is used as an extension edge.
- Each simple loop is traversed as a fixed number of times equal to the estimated multiplicity of the loop. The loop multiplicity is estimated as the closest integer to C_{loop}/C_{local}, where C_{loop} is the average read coverage of the loop edges, and C_{local} is the read coverage in the vicinity of the loop, i.e. the average read coverage of the last ReadLength nucleotides of edge e and first ReadLength nucleotides of edge e' (Fig. 4b).

Our benchmarking has demonstrated that the described modifications reduce the running time by an order of magnitude.

2.4 Scaffolding

2.4.1 Jumping over the coverage gaps

Existing genome assemblers are often complemented by scaffolding tools such as Bambus (Pop *et al.*, 2004), Opera (Gao *et al.*, 2011), SCARPA (Donmez and Brudno, 2013), SOPRA (Dayarian *et al.*, 2010) and SSPACE (Boetzer *et al.*, 2011) to improve the contiguity of the resulting assemblies. However, most of these scaffolding tools work with contigs rather than the assembly graphs and none of them uses the rectangle framework that exSPAnder utilizes. We thus added a rectangle-based scaffolder to exSPAnder and evaluated its performance.

Jumping from an out-tip to an in-tip. A coverage gap typically creates a *pair of tips* in the assembly graph [see Bankevich *et al.* (2012) for details] formed by an *out-tip* (an edge that ends in a vertex with out-degree 0) and an *in-tip* (an edge that starts in a vertex with in-degree 0). We further refer to these tips as *paired tips*. Since coverage gaps are usually short, read-pairs from the jumping libraries often span them. Below we describe how *ExtensionSearch* finds pairs of tips and uses them for scaffolding.

Let E be an extension path for a path P, such that E ends with an out-tip e and $\text{Length}(E) < I_{max}$ (Fig. 5a). To further extend path E, we search for an in-tip paired with e by considering a set of *P-supported edges*, i.e. all edges e' that have (P, e')-connecting

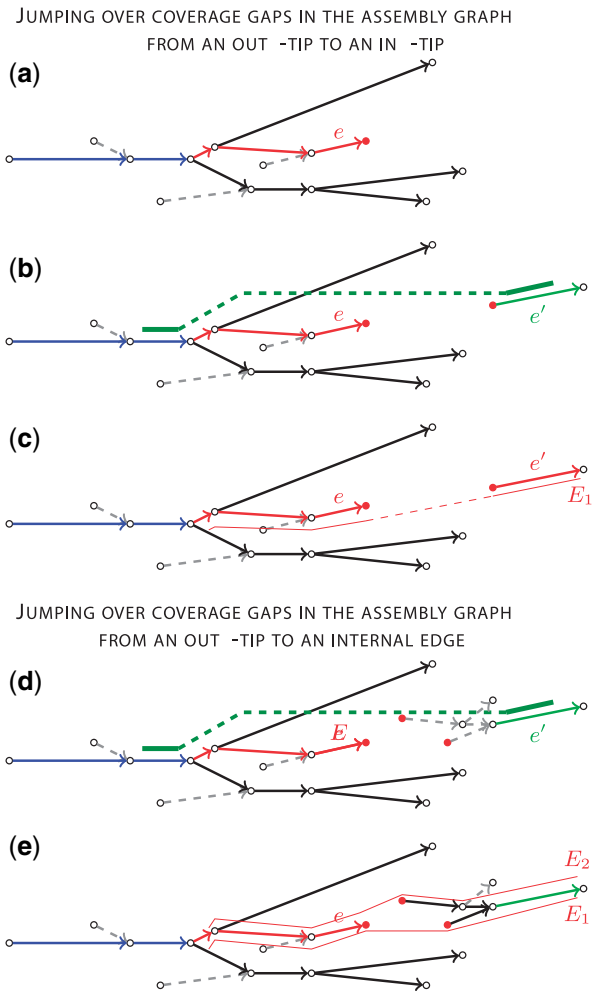

Fig. 5. Jumping over coverage gaps in the assembly graph from an out-tip to an in-tip: (**a**) an extension path E (marked red) ending with an out-tip e; (**b**) the only P-supported edge e' is an in-tip (marked green); (**c**) the path E is extended by an edge e' with a gap (red line). Jumping over coverage gaps in the assembly graph from an out-tip to an internal edge: (**d**) the only P-supported edge e' is not an in-tip (marked green) and (**e**) paths E_1 and E_2 that start with in-tips and contain P-supported edge e', both paths are considered as possible extensions for E (shown by red lines)

read-pairs. Since the set of P-supported edges typically contains many false short edges (due to high rate of chimeric read-pairs in the jumping libraries), we filter out all P-supported edges shorter than a certain threshold (default value is 500 bp).

Figure 5b presents a simple example when there is only one P-supported edge e' and this edge is an in-tip. We then consider e and e' as paired tips and extend the path E by edge e' with a gap (Fig. 5c). Since the size of a gap is difficult to estimate using a jumping library with large insert size variation, we assign a fixed length to each gap (100 bp by default). Once we extended the path E with an edge, we continue growing E using *ExtensionSearch* until its length exceeds $I_{\max}$.

Jumping from an out-tip to an internal edge. A more difficult (and more common) case is illustrated in Figure 5d, where there exists only one P-supported edge e' and this edge is not an in-tip. Such situations arise when the in-tip paired with e is either not P-supported or is shorter than 500 bp (and thus ignored). To grow the path E in such cases, we consider *all* in-tips in the assembly graph and extend them further using *ExtensionSearch* limiting the path length to $I_{\max} - \text{Length}(E)$. Afterward, we select only paths that contain edge e' and use them as potential extensions for E (Fig. 5e).

When assembling real datasets, we often encounter several P-supported edges longer than 500 bp. In this case, we apply the strategy described above, i.e. extend the paths starting from every in-tip and choose paths containing at least one P-supported edge. If some of the extension paths remain shorter than $I_{\max}$, we continue growing them with the *ExtensionSearch* algorithm.

However, growing paths from every in-tip can be extremely time-consuming, especially for single-cell datasets that often contain thousands of coverage gaps. To address this issue, we implement the described approach the other way around: instead of starting a path from every in-tip, we start growing paths from P-supported edges in the reverse direction and select only those that reach an in-tip. Since the number of long P-supported edges is typically much smaller than the number of all in-tips in the assembly graph in the case of single-cell assemblies, this implementation greatly reduces the running time of exSPAnder.

2.4.2 Jumping over complex subgraphs

Even after applying the described heuristics for run-time reduction, there are cases when the number of extension paths remains huge. Such cases are often associated with complex subgraphs of the assembly graph (typically formed by short repeats with high multiplicities). To keep the running time under control, exSPAnder stops growing a path if the number of its extension paths exceeds a threshold. The threshold is automatically calculated based on the number of edges in the assembly graph (e.g. the threshold value is set to 50 for graphs with $> 10\,000$ edges).

A better approach is to use jumping libraries and to 'jump over' complex subgraphs in the assembly graph (instead of generating all extension paths traversing these complex subgraphs). If the 'size' of a complex subgraph is smaller than the insert size, a path P may start on 'one side' of this subgraph and be connected with edges on the 'other side' of this subgraph, thus allowing us to jump over the entire subgraph and to continue growing P. The question is how to identify such complex subgraphs and jump over them.

To jump over complex subgraphs, we first identify a path P with unusually large number of extension paths. Afterward, we find all P-supported edges and order them using the rule: an edge e' follows an edge e if e' can be reached by *ExtensionSearch* within a certain distance starting from e. If all P-supported edges can be ordered into a *single* list, we extend path P by the first edge in this list (with a gap).

3 Results

To evaluate how availability of Nextera Mate Pair libraries affects the quality of assembly, we assembled several bacterial genomes using multicell datasets provided by Illumina (Nextera Mate Pair libraries only). In this article, we describe benchmarking (using seven different assemblers and three scaffolders) on *Meiothermus ruber str. 21T* (Tindall *et al.*, 2010) Nextera Mate Pair library and additionally, on *Escherichia coli st. K12 subst. MG1655* (Blattner *et al.*, 1997) single-cell dataset that contains both a short paired-end library and a long jumping library. Results for other Nextera Mate Pair libraries are presented in the Supplementary Material.

Since the assemblers we used for comparison were not designed for assembling Nextera Mate Pair libraries, we have conducted

Table 1. Comparison of the assemblies for the *M.ruber str. 21T* (Tindall *et al.*, 2010) (genome size 3.1 Mb) Nextera Mate Pair library

Assembler	*K*	NGA50	No.	Largest	No. mis	GF	No. *N*'s
ABySS	59	61	100	162	4	99.4	599
IDBA-UD	—	55	139	168	7	99.1	**16**
Opera	—	111	44	272	18	99.2	115
Ray	57	7	373	45	95	84.2	1834
SCARPA	—	129	61	389	**1**	99.2	**1**
SOAPdenovo	71	1	1449	411	244	92.8	36 360
SSPACE	—	305	14	757	6	99.4	**26**
Velvet	99	1716	11	**3011**	4	**99.7**	541
Velvet-SC	105	4	1099	21	18	98.3	0
SPAdes 3.0	—	1701	6	**2879**	3	**99.9**	**37**
SPAdes 3.5 ctg	—	1716	6	1716	2	**99.9**	**0**
SPAdes 3.5	—	**2892**	**5**	**2893**	2	**99.9**	**23**

Each assembly is represented by scaffolds if not indicated otherwise (marked with ctg). *K*, the *k*-mer size; NGA50, in kb; no., the total number of contigs/scaffolds longer than 500 bp; largest, for the length of the longest contig/scaffold assembled (in kb); no. mis is the number of misassemblies; GF, the percentage of genome mapped; No. *N*'s, the number of unspecified nucleotides per 100 kb. In each column, the best values are indicated in bold. Assemblies with very poor quality (e.g. low genome fraction, small NG50, relatively large number of misassemblies) are shown in gray and are ignored when selecting the best values for each metric. All metrics were computed using only contigs/scaffolds longer than 500 bp.

Table 2. Comparison of the assemblies for the *E.coli st. K12 subst. MG1655* (Blattner *et al.*, 1997) (genome size 4.7 Mb) single-cell dataset with paired-end and jumping libraries

Assembler	*K*	NGA50	No.	Largest	No. mis	GF	No. *N*'s
Only paired-end library							
ABySS	47	87	186	199	9	90.5	111
IDBA-UD	—	105	482	265	7	**95.0**	**0**
Opera	—	67	513	202	4	94.6	**7**
Ray	55	49	422	198	23	90.3	1475
SCARPA	—	84	486	209	3	**94.8**	**6**
SOAPdenovo	71	16	728	150	7	78.2	17
SSPACE	—	66	534	209	**2**	94.6	**0**
Velvet	81	22	**128**	178	3	62.3	10
Velvet-SC	55	21	634	135	4	92.7	**0**
SPAdes 3.0	—	117	453	**285**	3	**94.9**	**0**
SPAdes 3.5 ctg	—	117	459	**285**	4	**94.9**	**0**
SPAdes 3.5	—	**121**	452	**285**	4	**94.9**	**2**
Both paired-end and jumping libraries							
ABySS	47	87	186	199	**9**	90.5	111
ALLPATHS-LG	—	1	483	132	23	50.7	3801
IDBA-UD	—	105	557	416	16	95.0	**0**
Opera	—	71	388	202	28	94.6	359
Ray	55	95	361	281	28	92.7	427
SCARPA	—	132	413	630	29	94.8	505
SOAPdenovo	71	16	673	150	63	78.3	743
SSPACE	—	123	402	488	21	94.7	478
Velvet	81	24	**145**	178	17	67.5	330
Velvet-SC	55	—	1617	1	0	29.4	0
SPAdes 3.0	—	**357**	404	**1551**	**11**	**95.6**	108
SPAdes 3.5 ctg	—	319	414	653	**10**	**95.6**	**0**
SPAdes 3.5	—	**356**	392	693	**10**	**95.6**	**36**

Each assembly is represented by scaffolds if not indicated otherwise (marked with ctg). *K*, the *k*-mer size; NGA50, in kb; no., the total number of contigs/scaffolds longer than 500 bp; largest stands for the length of the longest contig/scaffold assembled (in kb); no. mis, the number of misassemblies; GF, the percentage of genome mapped; no. *N*'s, the number of unspecified nucleotides per 100 kb. In each column, the best values are indicated in bold. Assemblies with very poor quality (e.g. low genome fraction, small NG50, relatively large number of misassemblies) are shown in gray and are ignored when selecting the best values for each metric. All metrics were computed using only contigs/scaffolds longer than 500 bp.

additional optimization and selected the optimal *k*-mer sizes (with respect to N50) to ensure fair benchmarking with these assemblers. We ran ABySS 1.3.6 (Simpson *et al.*, 2009), Ray 2.0.0 (Boisvert *et al.*, 2010), SOAPdenovo 2.0.4 (Li *et al.*, 2010), Velvet 1.2.10 (Zerbino and Birney, 2008) and Velvet-SC (Chitsaz *et al.*, 2011) (based on Velvet 0.7.62, released on March 11, 2011) with the *k*-mer sizes 59, 57, 71, 99 and 105, respectively. Iterative de Bruijn graph assemblers IDBA-UD 1.1.1 (Peng *et al.*, 2012), SPAdes 3.0.0 (with the previous version of exSPAnder (Prjibelski *et al.*, 2014)] and SPAdes 3.5.0 (that implements the new algorithm described in this article) were run with the default parameters. In addition to scaffolds, we also provide information about contigs generated by SPAdes 3.5 (referred to as SPAdes 3.5 ctg). We have also included results for such popular scaffolders as Opera 2.0 (Gao *et al.*, 2011), SCARPA 0.241 (Donmez and Brudno, 2013) and SSPACE 3.0 (Boetzer *et al.*, 2011). To perform a fair comparison, we ran all scaffolders on the contigs that were assembled by SPAdes 3.5.0 using all data as single-end reads (discarding read-pair information).

The resulting assemblies were evaluated with QUAST 2.3 (Gurevich *et al.*, 2013) using standard metrics: NGA50 (NG50 corrected for assembly errors), the total number of scaffolds in the assembly, the size of the largest scaffold, the number of misassemblies, the fraction of genome covered and the number of uncalled bases (*N*) in the assembly.

Table 1 benchmarks various assemblers on *M.ruber* Nextera Mate Pair library (mean insert size 3.6 kb). Some of the assemblers used in the comparison produce rather inaccurate assemblies (e.g. 95 misassemblies for Ray and 244 misassemblies for SOAPdenovo). Also, some assemblers generate very large number of unspecified symbols N (ABySS, Ray, SOAPdenovo and Velvet). Interestingly, most assemblers showed rather unstable behavior with Nextera Mate Pair libraries with exception of IDBA-UD and SPAdes (originally developed as single-cell assemblers). SPAdes + exSPAnder assembles an almost complete *M.ruber* genome with less than 0.03% of unspecified nucleotides and the largest scaffold capturing more than 93% of the genome.

SPAdes generates similar high-quality assemblies on all Nextera Mate Pair libraries (see Supplementary Material) and often results in assemblies of very few contigs with the quality that approaches the quality of the assemblies from long Pacific Biosciences reads (Chin *et al.*, 2013). Thus, Nextera Mate Pair libraries provide a valuable low-cost trade-off when compared with the assemblies that use Pacific Biosciences reads.

In Table 2, we also present a comparison between selected tools on *E.coli* single-cell dataset with paired-end and jumping libraries. In addition, we ran ALLPATHS-LG (Gnerre *et al.*, 2011) (build 47561, released on September 15, 2013) with the default parameters using both libraries together. As Table 2 indicates SPAdes (both versions 3.0 and 3.5) and IDBA-UD outperform other tools on this dataset. However, when using both libraries together, SPAdes generates more accurate and continuous assemblies comparing to IDBA-UD and at the same time captures the largest genome fraction.

Acknowledgement

We would like to thank Ilya Chorny from Illumina for providing various datasets for Nextera Mate Pair libraries, which are now available at Illumina BaseSpace public repository (https://basespace.illumina.com).

Funding

This study was supported by the Russian Science Foundation [14-50-00069].

Conflict of Interest: none declared.

References

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

Blattner,F.R. *et al*. (1997) The complete genome sequence of *Escherichia coli* K-12. *Science*, **277**, 1453–1462.

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

Boisvert,S. *et al*. (2010) Ray: simultaneous assembly of reads from a mix of high-throughput sequencing technologies. *J. Comput. Biol.*, **17**, 1519–1533.

Bresler,M. *et al*. (2012) Telescoper: de novo assembly of highly repetitive regions. *Bioinformatics*, **28**, 311–317.

Chaisson,M. *et al*. (2009) De novo fragment assembly with short mate-paired reads: does the read length matter?. *Genome Res.*, **19**, 336.

Chin,C.-S. *et al*. (2013) Nonhybrid, finished microbial genome assemblies from long-read SMRT sequencing data. *Nat. Methods*, **10**, 563–569.

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

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. USA.*, **108**, 1513.

Gurevich,A. *et al*. (2013) QUAST: quality assessment tool for genome assemblies. *Bioinformatics*, **29**, 1072–1075.

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

Peng,Y. *et al*. (2012) IDBA-UD: A de novo assembler for single-cell and metagenomic sequencing data with highly uneven depth. *Bioinformatics*, **28**, 1–8.

Pevzner,P.A. *et al*. (2001) An Eulerian path approach to DNA fragment assembly. *Proc. Natl. Acad. Sci. USA.*, **98**, 9748–9753.

Pop,M. *et al*. (2004) Hierarchical scaffolding with *Bambus*. *Genome Res.*, **14**, 149–159.

Prjibelski,A.D. *et al*. (2014) ExSPAnder: a universal repeat resolver for DNA fragment assembly. *Bioinformatics*, **30**, i293–i301.

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

Tindall,B.J. *et al*. (2010) Complete genome sequence of *Meiothermus ruber* type strain (21T). *Stand. Genomic Sci.*, **3**, 26–36.

Vyahhi,N. *et al*. (2012) From de Bruijn graphs to rectangle graphs for genome assembly. In: Raphael,B. and Tang,J. (eds) *Workshop on Algorithms in Bioinformatics* 2012, volume 7534 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 200–212.

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

Zhu,X. *et al*. (2014) PERGA: a paired-end read guided de novo assembler for extending contigs using SVM and look ahead approach. *PLoS One*, **9**, e114253.