Localized genome assembly from reads to scaffolds: practical traversal of the paired string graph

Rayan Chikhi, Dominique Lavenier

Localized genome assembly from reads to scaffolds: practical traversal of the paired string graph

Rayan Chikhi and Dominique Lavenier

Computer Science department, ENS Cachan/IRISA, 35042 Rennes, France

Abstract. Next-generation de novo short reads assemblers typically use the following strategy: (1) assemble unpaired reads using heuristics leading to contigs; (2) order contigs from paired reads information to produce scaffolds. We propose to unify these two steps by introducing localized assembly: direct construction of scaffolds from reads. To this end, the paired string graph structure is introduced, along with a formal framework for building scaffolds as paths of reads. This framework leads to the design of a novel greedy algorithm for memory-efficient, parallel assembly of paired reads. A prototype implementation of the algorithm has been developed and applied to the assembly of simulated and experimental short reads. Our experiments show that our methods yields longer scaffolds than recent assemblers, and is capable of assembling diploid genomes significantly better than other greedy methods.

1 Introduction

De novo assembly of short reads consists in reconstructing a genome sequence given a set of short subsequences (*reads*) obtained from DNA sequencing. In practice, the original sequence cannot be retrieved unambiguously because a genome contains repetitions longer than the reads. Hence, one aims at finding a reasonable approximation of the sequence as a set of longer gap-less subsequences (*contigs*) or gapped subsequences (*scaffolds*). Over the last decade, three different genome assembly approaches have been adopted. Two of them are graph-based. The *string graph* method is based on a graph containing all the overlaps between reads [22]. The *de Bruijn graph* approach is based on the graph of all the k-length substrings of reads [24,21,13]. The third approach performs greedy extension of contigs using an ad-hoc structures [3,23]. Even producing an approximation of the genome is a computationally difficult task. For assembly of

human-sized genomes using short reads (< 100 bp), current state of the art implementations (using de Bruijn graphs) require hundreds of gigabases of memory and several CPU weeks of computation [13]. For more details concerning assembly approaches, refer to a recent survey [15].

Every next-generation sequencing technology is now able to massively produce *paired reads* separated by a known approximate distance (*insert size*). This information is highly valuable for re-sequencing projects as it enables better mapping coverage, especially with long inserts [6]. It also permits better resolution of repeats shorter than the insert size for de novo assembly. To illustrate the difference between single-end and paired assembly, an analogy with jigsaws can be made: consider n jigsaw pieces where each piece is linked to another piece by a string of finite length. The problem is to decide whether these pieces exactly fit into a $\sqrt{n} \times \sqrt{n}$ box with the additional constraint that every string must be tightened. In other words, pairing information indicates how far apart two pieces are in the solution. It can be shown that the paired jigsaw problem is also NP-complete. The reduction consists of splitting each classical jigsaw piece into 12 paired pieces.

Practically, pairs add more structure to the assembly problem, by indicating the relative position of reads on the genome. While most current assemblers use pairing information to improve assembly quality, they rely on single-end assembly beforehand. For instance, recent implementations of de Bruijn graph assembly methods use paired reads to simplify the graph once it is fully constructed and simplified from single-end reads [9,24,13]. The process of improving an existing assembly using paired reads is called *scaffolding.* The scaffolding problem is NP-complete and several heuristics have been proposed [18,11]. However, it may appear unsatisfactory to perform paired assembly using graph simplification or scaffolding, as such approach requires to solve unpaired assembly beforehand, which essentially ignores helpful pairing constraints.

Previous research has explored the benefits of using paired-end reads during contigs construction. The Arachne assembler searches for pairs of paired Sanger reads where both mates overlap to construct contigs [2]. The Shorty assembler uses pairing information to greedily construct contigs from paired reads anchored to long

reads [10]. PE-Assembler extends contigs greedily and attempts to resolve ambiguous extensions using paired reads anchored nearby [1]. Medvedev et al. recently introduced the paired de Bruijn formalism, which incorporates pairing information in the de Bruijn graph [14]. Donmez et al. also recently proposed an approach to transform a string graph into a mate-pairs graph [7]. Each of these approaches aim to resolve repeats when constructing contigs. Our approach is essentially different as it uses paired reads for direct scaffold construction. One main advantage is that missing read overlaps (possibly due to sequencing artefacts, such as coverage gaps or localized errors) can be represented by gaps in scaffolds, whereas they would necessarily interrupt contigs. Note that all methods, including ours, do not implement mechanisms to resolve repetitions longer than the insert size.

In the next section, assembly of paired reads is formalized using the paired string graph representation. It is shown that scaffolds correspond to paths in the graph under ideal sequencing conditions. The definition of these paths is then refined to account for sequencing errors and biological variants. In section 3, a prototype implementation of path construction is applied to simulated and experimental sequencing data. A comparison is made with two other popular assemblers, using relevant assembly quality metrics.

2 Paired string graph and non-branching paths

2.1 Paired assembly problem

The paired string graph is defined as an extension of the classical string graph [16] over a set of paired reads $R_1 \times R_2$. Two reads $(r, r') \in R_1 \cup R_2$ are said to *k-overlap* if a suffix of r matches a prefix of r' exactly over k characters. The *paired string graph* $PG^k(R_1 \times R_2)$ is defined as a directed graph by assigning a vertex to each read in $R_1 \cup R_2$. An edge $r \rightarrow r'$ is created between two reads if r k-overlaps r' (*overlap edge*). A special type of edge $r \dashrightarrow r'$ is created if (r, r') is a paired read (*paired edge*). Classical string graph transformations are applied: reads that are substrings of other reads, and transitively redundant overlap edges are discarded (paired edges are ignored during this step). No transitive reduction is performed for

paired edges. For instance, consider the sequence $S = ab\mathbf{cd}ef\mathbf{cd}gh$ and perfect sequencing with insert of length 6 and paired reads of length 2. The paired string graph of these reads is drawn in Figure 1.

A *mixed path* in the paired string graph is a succession of vertices linked by either overlap edges or paired edges, e.g. $r_1 \rightarrow r_2 \dashrightarrow r_3 \rightarrow r_4$. *A path-string* is a string corresponding to the concatenation of nodes strings along a mixed path. The path-string is formed by the following rules: after an overlap edge, the string is appended with the concatenation of both nodes strings with their overlap repeated only once; after a paired edge, the string is appended with a gap corresponding to the paired insert size. In Figure 1, the path-string of $p = ab \rightarrow bc \dashrightarrow fc$ is $abc\Diamond^2 fc$, where $\Diamond$ is a single-character gap.

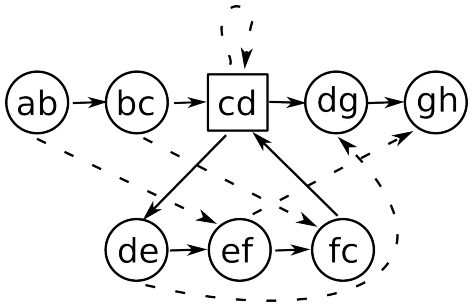

Fig. 1. Example of a paired string graph from paired reads (insert size of 6) covering the sequence $S = ab\mathbf{cde}f\mathbf{cd}gh$. Dashes edges represent paired links and regular edges represent 1-overlaps between reads.

Similarly to the Assembly Problem (AP) [17], the *Paired Assembly Problem* can be defined as a constrained flavor of AP. The Paired Assembly Problem consists in finding a path that visits each node at least once (generalized Hamiltonian path) in $PG^k(R_1 \times R_2)$, and corresponds to a path-string s such that (1) the length of s is minimized and (2) for every pair (r, r') in $R_1 \times R_2$, the distance between r and r' in s matches the paired insert size. Note that a solution is necessarily a contig. Similarly to AP, this problem can also be shown to be NP-hard. The following section focuses on constructing a collection of sub-paths (possibly scaffolds) that approximate a solution.

2.2 Non-branching paths in the ideal case

Scaffolds can be directly constructed from the graph by following special types of mixed paths. To illustrate this, we first assume unrealistic sequencing conditions: error-free reads, perfect coverage and exact insert size (these will be relaxed in the next section). A mixed path in $PG^k(R_1 \times R_2)$ is a *non-branching path* (NBP) if each node, except the first and the last, has in-degree of 1 in the graph with respect to the corresponding in-edge type in the path, and out-degree of 1 corresponding to the out-edge type. In traditional assembly heuristics, a contig can be represented as a NBP where each edge is an overlap edge (*simple path*). For example, maximal-length contigs from the graph in Figure 1 are

$$\{ab \rightarrow bc \rightarrow cd, cd \rightarrow de \rightarrow ef \rightarrow fc \rightarrow cd, cd \rightarrow dg \rightarrow gh\}.$$

In contrast, a non-trivial non-branching path is

$$\{ab \dashrightarrow ef \dashrightarrow gh\},$$

where the path-string $(ab\Diamond^2 ef\Diamond^2 gh)$ is a scaffold which covers the whole string. Under ideal sequencing conditions, non-branching paths immediately correspond to valid scaffolds. One can also consider *in-(resp. out-) branching paths*, for which only out- (resp. in-) degree of nodes in the path with respect to the corresponding edge type is 1. By similar reasoning, it can be shown that such paths also spell valid scaffolds.

2.3 Practical non-branching paths

In actual sequencing, we distinguish two situations: undetected paired branching and additional overlap branching. Previously, paired branching was always detected because of perfect coverage and exact insert size. Now, it is no longer sufficient for a node to have an unique paired edge in order to unambiguously extend a scaffold. Weaker conditions can be formulated to detect the absence of paired branching, given imperfect coverage and variable insert size. First, assume that the insert size deviation is bounded by a constant i. Second, consider a simple path p of length $2i + 1$, and let n be the central node (p_{i+1}).

Property 1. A paired edge $n \dashrightarrow n'$ is considered to satisfy the non-branching condition if the sub-graph induced by the opposite mates of nodes in p is a simple path p' of central node $n' = p'_{\lfloor \frac{|p'|}{2} \rfloor}$.

In other words, it is possible to detect that p' is the only genomic region which appears at approximately an insert distance further than p. The original definition of non-branching paths can then be extended to include this condition in place of the paired degree condition.

Furthermore, sequencing errors and biological variants introduce additional branching in the graph. The branching structures are referred as *bubbles* (multiple paths that starts and ends at the same nodes) and *tips* (short interrupted paths) [24]. Graph-based assembly algorithms typically remove bubbles and tips after the whole graph is constructed. Here, the bubble detection technique presented in [24] is adapted to also detect tips. As these structures are short, one can set a maximal length $d > 0$ for paths within them. A general characterization of these structure can be made in terms of sub-graph traversal. Observe that both structures form sub-graphs which start at a single node, and paths which are not interrupted converge after a certain length.

Property 2. Given a variant sub-graph, the breadth-first tree constructed from the start node has a single node of depth d.

Non-branching paths are extended to permit traversal in short branching sub-graphs through overlap edges. Figure 2 illustrates both properties. In summary, we define practical non-branching paths (PNBP) as follows:

- for path nodes $n \dashrightarrow m$ linked by a paired edge, both n and m are middle nodes of simple paths of length $2i + 1$ for which Property 1 is verified.
- for path nodes $n \rightarrow m$ linked by an overlap edge, either the overlap out-degree of n and the overlap in-degree of m are both 1, or $n \rightarrow m$ is part of a sub-graph which satisfies Property 2.

Note that setting $i = 0$ and $d = 1$ corresponds to the original definition of non-branching paths. Practical in-branching (resp. out-branching) paths are defined similarly, except that Property 1 only needs to be verified for n (resp. m).

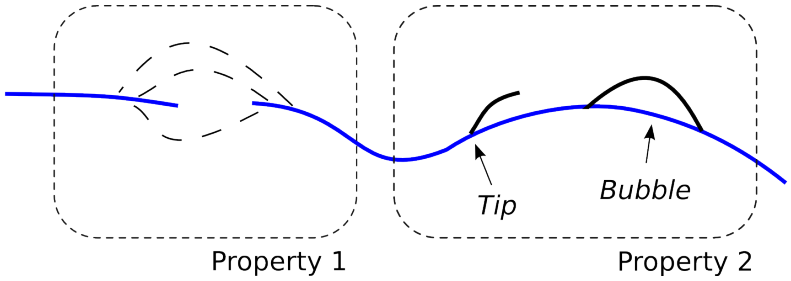

Fig. 2. Practical non-branching path traversal (blue line) of a paired string sub-graph. Thick lines represent paths of overlap edges. Dashed lines represent paired edges between reads. Property 1 is used to traverse a gap, as paired reads link together two simple paths. Property 2 is used to traversal small branching regions (a tip and a bubble).

2.4 Localized, parallel assembly

For large genomes, constructing the classical string graph is a memory-intensive operation. This issue also applies to paired string graphs, as they contain strictly more information. Two possible solutions are considered to reduce memory usage. A compressed FM-index [8] of the reads reduces the memory usage of the string graph [20]. This approach could be extended to include paired edges, computed dynamically from indexed paired reads to avoid memory overhead.

Another solution is to perform localized assembly, without initial construction of the whole paired string graph. The key property of practical non-branching paths is that only a small sub-graph needs to be explored for each path. We propose to take advantage of this property to perform assembly both locally and in parallel. Concretely, a greedy assembly algorithm can start from any read and construct a sub-graph of the paired string graph on the fly, by following a PNBP strategy. This is equivalent to splitting the complete paired string graph into disjoint sub-graphs, each sub-graph corresponds to exactly one scaffold. This approach induces a memory overhead due to the parallel construction of sub-graphs, however only a constant number of scaffolds is assembled in parallel at any given time. To construct each sub-graph, overlaps between paired reads and pairing information need to be accessed efficiently. Hence, it is required that a complete index of the reads resides in memory. However, such index occupies less memory than a string graph. Provided that each

worker can access the full index, embarrassingly parallel construction of scaffolds can be achieved.

While it is technically a greedy algorithm, such approach overcomes the shortcomings of classical greedy algorithms. Except for Taipan [19], which does not support paired reads nor parallel assembly, most greedy assemblers do not construct a graph to solve local extension ambiguities [4,1,3]. As a consequence, greedy assemblers stop contigs extension at small biological variations or sequencing artifacts. Localized graph-based assembly using Property 2 overcomes this problem by explicitly performing traversal of such structures.

3 Results

We developed a prototype assembler called Monument based on local construction of practical in-branching paths. The prototype is implemented in the Python language with C++ extensions for critical parts. For memory efficiency, a kmer-based reads indexing structure is used. Specifically, the indexing procedure minimizes the number of reads referenced by each kmer, while still maintaining branching information. For practical in-branching paths, the maximal graph depth d for genomic variants is set such that any path has genomic length less than $10 + k$. The insert size deviation i is set to half the value of the insert size, which is a very conservative deviation with respect to actual paired-end data. To ensure fair comparison with other methods, we implemented a naive gap-closing procedure which fill scaffolds gaps with any overlap path satisfying insert size constraints.

Two short reads assemblers based on de Bruijn graphs are compared with this prototype. The Velvet assembler (version 1.1.03) uses graph simplification heuristics [24]. The Ray assembler (version 1.3.0) implements a greedy traversal strategy [3]. The assemblers were run with default parameters and $k = 23$. By setting a similar kmer size, all assemblers, including ours, virtually explore the same de Bruijn graph.

We first compared assemblers on experimental Illumina short paired reads from *E. coli* (SRA SRX000429). This dataset (Dataset 1) contains 10 million paired reads of length 36 bp and insert size 200 bp. We then investigated the ability of our method to assemble

diploid genomes. To this end, we simulated 3 million paired reads of a diploid genome based on the *E. coli* sequence (Dataset 2). The wgsim paired reads simulator was used with default parameters [12], producing 75 bp reads (500 bp inserts) with simulated sequencing errors. Assembly results for the datasets are shown in Table 1. To understand why Ray has difficulties assembling the second dataset, we simulated a third dataset of reads, similar to Dataset 2 but without variants. This time, Ray obtains a scaffold N50 of 89.4 Kbp and largest scaffold of length 268.5 Kbp. This experiment confirms that mechanisms for biological variations traversal, such as PNBPs, are a key requirement for greedy assemblers.

Dataset	Software	Contig N50 (Kbp)	Scaffold N50 (Kbp)	Longest scaffold (Kbp)	Coverage (%)	Accuracy (%)
Experimental (1)	Monument	38.0	101.8	236.0	96.4	96.7
	Velvet	26.3	95.3	267.9	96.9	99.1
	Ray	69.5	87.3	174.4	97.4	98.4
Simulated with variants (2)	Monument	113.3	134.1	340.5	91.0	95.0
	Velvet	30.8	132.6	327.2	87.9	92.3
	Ray	10.2	10.2	41.2	89.2	100.0

Table 1. Quality of the assemblies of simulated and experimental paired-end reads from *E.coli* using Velvet, Ray and our prototype (Monument). The N50 metric measures the length of the smallest element of the set of largest scaffolds (resp. contigs) which cover at least 50% of the assembly. Coverage and accuracy of scaffolds are assessed using evaluation tools from Allpaths [5]. Specifically, scaffolds are divided into chunks of size less than 10kb. Considering the high coverage of the datasets, each chunk is considered to be valid if it aligns with more than 99% identity to the reference genome (alignment with undertermined nucleotides are considered valid).

We recorded execution time and memory usage during the assembly of the experimental dataset. The size of the paired reads index is 0.4 GB and peak memory usage during assembly is 0.6 GB. Velvet and Ray have peak memory usage of 2.4 GB and 3.2 GB respectively. However, Ray can distribute its indexing structure on a cluster. Using 6 threads, our implementation completed the assembly in 7 minutes, Velvet in 8 minutes and Ray in 16 minutes.

Our implementation can also assemble a scaffold around a specified genomic region, i.e. perform targeted assembly. This is of particular interest as new targeted assembly methods (TASR and Mapsembler, both unpublished) only produce contigs. Targeted assembly with our prototype is also very fast: one scaffold is assembled in a few seconds. However, contrary to targeted assemblers, the prototype requires the complete reads index to reside in memory.

4 Discussion

In summary, a new *de novo* assembly framework is formulated by introducing paired string graphs. The novelty of this framework resides in its ability to perform localized assembly of scaffolds. Prior to this work, scaffolds were constructed from an ordering of contigs, requiring a complete assembly of contigs to be known beforehand. We show that it is possible to assemble scaffolds locally around a genomic region by following non-branching paths greedily. This approach allows to design the first localized assembly algorithm which directly constructs scaffolds from reads.

Compared to other greedy approaches, our approach takes into account biological variants. Hence, it does not suffer from degraded contiguity with diploid genomes. Preliminary benchmark results on simulated and experimental datasets indicate that this method yields longer scaffolds than two leading short reads assemblers. We conjecture that scaffolders implemented in popular assemblers do not take full advantage of the whole contigs graph, as our greedy traversal obtains comparable results. Additionnally, practical benefits of this algorithm are twofold. It is embarrassingly parallelizable, as scaffolds can be constructed independently. It also does not require a large graph to be stored in memory, a small graph is constructed for each scaffold.

A natural future direction for this work is to compare the prototype with more existing tools, especially string graphs implementations, on larger datasets. Two other aspects should also be considered: (1) gap-closing in scaffolds is a key step for obtaining long contigs. Most complex repeats were not resolved by our simple path-finding procedure, hence a more elaborate algorithm is needed. (2) Incorporating mate-pairs with long inserts in genomic graphs is still

an unaddressed challenge in the literature. These reads are produced with higher insert size variability and lower coverage than paired-end reads. Mate-pairs cannot be used in our current framework, because Property 1 almost never holds for such data. An immediate solution would be to perform re-scaffolding of scaffolds using mate-pairs links.

As short read sequencing is progressively shifting towards longer reads (over 100 bp), the landscape of assembly software has to adapt to high-coverage, longer reads. Specifically, de Bruijn graph implementations appear to be unable to assemble long reads with quality comparable to string graph implementations. In contrast, string graph-based methods are limited to assembly of low-volume datasets because of memory constraints. We believe that our methodology will lead to software able to assemble both short and long reads at any coverage without sacrificing running time or results quality.

Acknowledgements

The authors are grateful to J. Nicolas for helpful discussions. This work benefited from the ANR grant associated with the MAPPI project (2010-2014).

References

1. Ariyaratne, P.N., Sung, W.: PE-Assembler: de novo assembler using short paired-end reads. Bioinformatics (Dec 2010)
2. Batzoglou, S., Jaffe, D.B., Stanley, K., Butler, J., Gnerre, S., Mauceli, E., Berger, B., Mesirov, J.P., Lander, E.S.: ARACHNE: a whole-genome shotgun assembler. Genome research 12(1), 177 (2002)
3. Boisvert, S., Laviolette, F., Corbeil, J.: Ray: Simultaneous assembly of reads from a mix of High-Throughput sequencing technologies. Journal of Computational Biology pp. 3389–3402 (2010)
4. Bryant, D., Wong, W., Mockler, T.: QSRA – a quality-value guided de novo short read assembler. BMC bioinformatics 10(1), 69 (2009)
5. Butler, J., MacCallum, I., Kleber, M., Shlyakhter, I.A., Belmonte, M.K., Lander, E.S., Nusbaum, C., Jaffe, D.B.: ALLPATHS: de novo assembly of whole-genome shotgun microreads. Genome Research 18(5), 810–820 (2008), http://genome.cshlp.org/content/18/5/810.abstract
6. Chikhi, R., Lavenier, D.: Paired-end read length lower bounds for genome resequencing. BMC Bioinformatics 10(Suppl 13), O2 (2009)
7. Donmez, N., Brudno, M.: Hapsembler: An assembler for highly polymorphic genomes. In: Research in Computational Molecular Biology. pp. 38–52. Springer (2011)

8. Ferragina, P., Manzini, G.: Indexing compressed text. Journal of the ACM (JACM) 52(4), 552–581 (2005)
9. Gnerre, S., MacCallum, I., Przybylski, D., Ribeiro, F.J., Burton, J.N., Walker, B.J., Sharpe, T., Hall, G., Shea, T.P., Sykes, S., Berlin, A.M., Aird, D., Costello, M., Daza, R., Williams, L., Nicol, R., Gnirke, A., Nusbaum, C., Lander, E.S., Jaffe, D.B.: High-quality draft assemblies of mammalian genomes from massively parallel sequence data. Proceedings of the National Academy of Sciences 108(4), 1513–1518 (2011), http://www.pnas.org/content/108/4/1513.abstract
10. Hossain, M., Azimi, N., Skiena, S.: Crystallizing short-read assemblies around seeds. BMC Bioinformatics 10(Suppl 1), S16 (2009), http://www.biomedcentral.com/1471-2105/10/S1/S16
11. Huson, D.H., Reinert, K., Myers, E.W.: The greedy path-merging algorithm for contig scaffolding. Journal of the ACM (JACM) 49(5), 603–615 (2002)
12. Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G., Abecasis, G., Durbin, R.: The sequence alignment/map format and SAMtools. Bioinformatics 25(16), 2078 (2009)
13. Li, R., Zhu, H., Ruan, J., Qian, W., Fang, X., Shi, Z., Li, Y., Li, S., Shan, G., Kristiansen, K., Li, S., Yang, H., Wang, J., Wang, J.: De novo assembly of human genomes with massively parallel short read sequencing. Genome Research 20(2), 265–272 (2010), http://genome.cshlp.org/content/20/2/265.abstract
14. Medvedev, P., Pham, S., Chaisson, M., Tesler, G., Pevzner, P.: Paired de bruijn graphs: A novel approach for incorporating mate pair information into genome assemblers. In: Research in Computational Molecular Biology. pp. 238–251. Springer (2011)
15. Miller, J.R., Koren, S., Sutton, G.: Assembly algorithms for next-generation sequencing data. Genomics (2010)
16. Myers, E.W.: Toward simplifying and accurately formulating fragment assembly. Journal of Computational Biology 2(2), 275–290 (1995)
17. Nagarajan, N., Pop, M.: Parametric complexity of sequence assembly: Theory and applications to next generation sequencing. Journal of Computational Biology 16(7), 897–908 (2009)
18. Pop, M., Kosack, D.S., Salzberg, S.L.: Hierarchical scaffolding with bambus. Genome Research 14(1), 149–159 (2004), http://genome.cshlp.org/content/14/1/149.abstract
19. Schmidt, B., Sinha, R., Beresford-Smith, B., Puglisi, S.J.: A fast hybrid short read fragment assembly algorithm. Bioinformatics 25(17), 2279 (2009)
20. Simpson, J.T., Durbin, R.: Efficient construction of an assembly string graph using the FM-index. Bioinformatics 26(12), i367 (2010)
21. Simpson, J., Wong, K., Jackman, S., Schein, J., Jones, S., Birol, İ.: ABySS: A parallel assembler for short read sequence data. Genome Research 19(6), 1117 (2009)
22. Sutton, G., Miller, J.R., Delcher, A.L., Koren, S., Venter, E., Walenz, B.P., Brownley, A., Johnson, J., Li, K., Mobarry, C.: Aggressive assembly of pyrosequencing reads with mates. Bioinformatics 24(24), 2818–2824 (2008), http://bioinformatics.oxfordjournals.org/cgi/content/abstract/24/24/2818
23. Warren, R.L., Sutton, G.G., Jones, S.J.M., Holt, R.A.: Assembling millions of short DNA sequences using SSAKE. Bioinformatics 23(4), 500–501 (2007), http://bioinformatics.oxfordjournals.org/cgi/content/abstract/23/4/500
24. Zerbino, D.R., Birney, E.: Velvet: Algorithms for de novo short read assembly using de bruijn graphs. Genome Research 18(5), 821–829 (2008), http://genome.cshlp.org/content/18/5/821.abstract