Yet another de novo genome assembler

1st Robert Vaser
dept. Electronic Systems and Information Processing
University of Zagreb
Faculty of Electrical Engineering and Computing
Zagreb, Croatia
robert.vaser@fer.hr

2nd Mile Šikić
dept. Electronic Systems and Information Processing
University of Zagreb
Faculty of Electrical Engineering and Computing
Zagreb, Croatia
dept. Computational Biology and Data Science
*Genome Institute of Singapore, A*STAR*
Singapore
mile.sikic@fer.hr

***Abstract*—Advances in sequencing technologies have pushed the limits of genome assemblies beyond imagination. The sheer amount of long read data that is being generated enables the assembly for even the largest and most complex organism for which efficient algorithms are needed. We present a new tool, called Ra, for de novo genome assembly of long uncorrected reads. It is a fast and memory friendly assembler based on sequence classification and assembly graphs, developed with large genomes in mind. It is freely available at https://github.com/lbcb-sci/ra.**

***Index Terms*—de novo genome assembly, long reads, read classification**

I. Introduction

The pace of improvement in sequencing technologies is staggering. From modest fragments up to a thousand nucleotides obtained with the first two generations of sequencing, the read length increased manifold after just a few decades, but with a setback in accuracy. Nonetheless, the increase in maximal sequencing length facilitated significant advances in contiguity of genome assemblies. Both leaders of the third generation of sequencing, Oxford Nanopore Technologies (ONT) and Pacific Biosciences (PB), are continuously improving their methods and throughput. Their novel protocols enable the generation of ultra-long [1] or highly-accurate reads [2], which will mitigate the assembly problem even further. However, de novo assemblies for larger genomes are still fragmented and substantial amount of computational resources is needed to acquire them.

There is a vast amount of available genome assemblers today. Some are specialized for short reads, others for long, and the middle balances the advantages and disadvantages of both sides. Almost all employ graph based techniques to retrieve contiguous chains of sequenced reads, followed by a polishing step to get rid of the remaining sequencing errors. Short read assemblers are successors of the De Bruijn graph approach [3], while long read assemblers compute pairwise overlaps between all reads in order to build the string graph [4] or some of its variations, like the best overlap graph [5] and the assembly graph [6]. Portion of the research shifted to the generalization of de Bruijn graphs to make them more resilient to error-ridden third generation data. Presence or absence of read error-correction prior the assembly is another criterion by which the assemblers distinguish themselves amongst others. A list of available tools for long read assembly and their characteristics can be observed in Table 1.

This work has been supported in part by the Croatian Science Foundation under the project Single genome and metagenome assembly (IP-2018-01-5886), and in part by the European Regional Development Fund under the grant KK.01.1.1.01.0009 (DATACROSS). In addition, M.Š. is partly supported by funding from A*STAR, Singapore.

In this paper we present yet another approach to genome assembly of long reads, incorporated in a tool called Ra (short for Rapid Assembler), which is based on the Overlap-Layout-Consensus paradigm (OLC) and is a mixture of well established concepts developed in a memory friendly manner.

II. Methods

Ra uses pairwise overlaps generated by minimap2 [12] for a given set of raw sequences to build an assembly graph, a directed graph that is both Watson-Crick complete and containment free [6]. As a preprocessing step, it trims sequence adapters, purges chimeric sequences and removes false overlaps induced by repetitive genomic regions located at sequence ends. This is achieved by examining their pile-o-grams, which are produced directly from pairwise overlaps [10]. After graph construction, Ra follows the default graph simplification path, i.e. transitive reduction, tip removal and bubble popping. Leftover tangles are resolved by cutting short overlaps. Linear paths of the assembly graph are extracted and passed to the consensus module Racon [13] to iteratively increase the accuracy of the reconstructed genome. Further-

TABLE I
Tools for de novo genome assembly of long reads

Tool	Approach	Reference
Canu	Error correction $\rightarrow$ Bogart	[7]
Falcon	Error correction $\rightarrow$ String graph	[8]
Flye	ABruijn graph	[9]
Hinge	Repeat-aware BOG	[10]
Miniasm	Assembly graph	[6]
Wtdbg2	Fuzzy Bruijn graph	[11]

Algorithm 1 Ra algorithm for de novo genome assembly

```
Input: Set of long and noisy sequences $T$ obtained with third generation of sequencing. Optionally, a set of short and accurate sequences $S$ obtained with second generation of sequencing.
Output: Set of assembled and polished unitigs $A$.
1:  procedure Ra($T$, $[S]$)
2:      $O \leftarrow$ Minimap2($T$, $T$)                                ▷ [12]
3:      $P \leftarrow$ TransformOverlaps($O$)        ▷ Create pile-o-grams
4:      for all $p \in P$ do
5:          Trim($p$)
6:          FindCoverageSlopes($p$)
7:      RemoveContainedSequences($P$, $O$)
8:      $C \leftarrow$ FindConnectedComponents($P$, $O$)                ▷ [14]
9:      for all $c \in C$ do
10:         $m \leftarrow$ FindCoverageMedian($c$)
11:         for all $p \in c$ do
12:             AnnotateChimericRegions($p$, $m$)
13:     ▷ Retain the longest non-chimeric part of uncontained sequences
14:     $U \leftarrow$ TransformSequences($P$, $T$)
15:     $Q \leftarrow$ Minimap2($U$, $T$, $f \leftarrow 10^{-5}$)
16:     $P \leftarrow$ TransformOverlaps($Q$)   ▷ Recreate pile-o-grams
17:     for all $c \in C$ do
18:         $m \leftarrow$ FindCoverageMedian($c$)
19:         for all $p \in c$ do
20:             AnnotateRepetitiveRegions($p$, $m$)
21:     ▷ Use repeat annotations to identify false overlaps
22:     RemoveFalseOverlaps($O$, $P$)
23:     $G \leftarrow$ CreateAssemblyGraph($U$, $O$)                     ▷ [6]
24:     RemoveTransitiveEdges($G$)                                ▷ [4]
25:     repeat
26:         $G' \leftarrow G$
27:         RemoveTips($G$)
28:         RemoveBubbles($G$)
29:     until $G' = G$
30:     RemoveLongEdges($G$)                                      ▷ [6]
31:     $A \leftarrow$ CreateUnitigs($G$)
32:     $R \leftarrow [T, T, S]$
33:     for all $r \in R$ do
34:         $O \leftarrow$ Minimap2($A$, $r$)
35:         $A \leftarrow$ Racon($r$, $O$, $A$)                          ▷ [13]
36:     return $A$
```

more, if second generation sequencing data is available, it can be used to further increase the accuracy of the assembly. Pseudocode of the whole Ra pipeline can be seen in Algorithm 1.

A. Preprocessing

Pile-o-grams are an impressive tool to dive deeper into the assembly problem, i.e. identification of sequence types. They can be created by stacking all pairwise overlaps of a sequence on top of each other. Summing up the number of overlaps covering each base yields a one-dimensional signal that has a characteristic outline depending on the sequence type. Sequences that uniquely and fully map to the sequenced genome should have almost uniform coverage across their length as shown in Fig. 1. Others have noticeable fluctuations in the signal. Chimeric sequences are sequencing artefacts consisting of multiple parts which are arranged in a way that is absent in the genome. Such signals mostly have a sharp decline in coverage which is followed with a sharp increase, between each of the connected parts (Fig. 2a). There are also cases when those parts overlap a certain amount, causing a spike that reminds of a Dirac delta function (Fig. 2b). Sequences containing a repetitive region at either of the ends can spawn false suffix-prefix overlaps. Those regions can be identified as an increase in coverage in form of a step function (Fig. 2c).

Hinge uses pile-o-grams to detect chimeric sequences and annotate repetitive genomic regions by calculating coverage gradients [10]. Chimeric sequences are truncated to the longest non-chimeric part. On the other hand, repeat annotations are used to find sequences that do not fully bridge certain repeats to allow some of them multiple overlaps in an otherwise best overlap graph, and later use this information for repeat resolution [10].

In the first stage Ra implements a similar approach. All pairwise overlaps are loaded in blocks and immediately transformed into pile-o-grams to decrease the memory footprint.

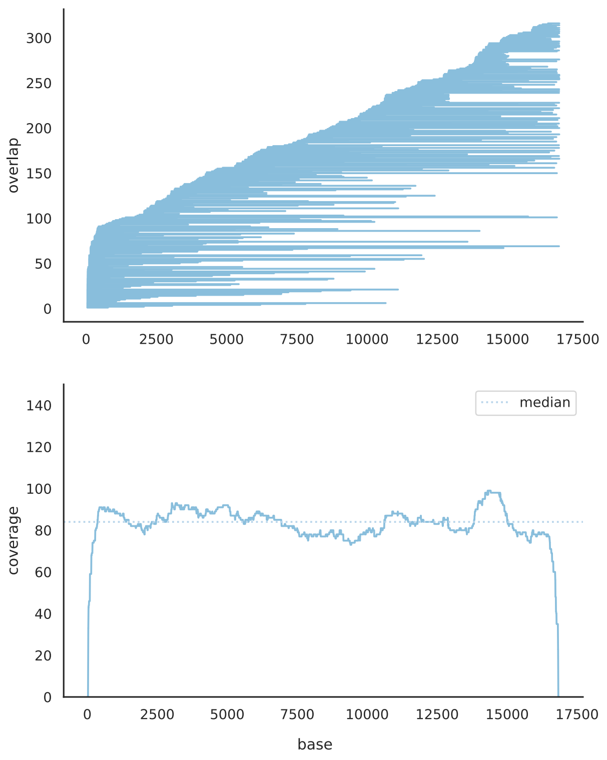

Fig. 1. Pile-o-gram that is almost uniform. It was obtained by adding up the number of overlaps covering each base in a sequence which can be uniquely mapped to the reference genome.

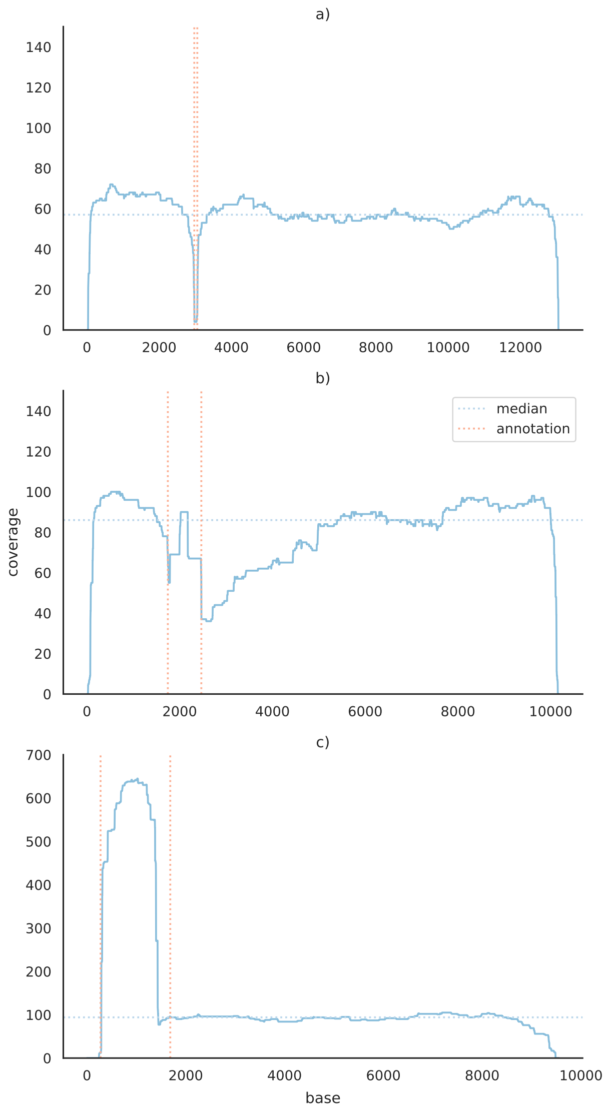

Fig. 2. Pile-o-gram representatives for other sequence types with highlighted regions of interest. Chimeric sequences, those that partially map to different regions of the genome, can be seen in subfigures a) and b). The pile-o-gram in a) has a rift denoting that there is almost no overlaps covering this part of the sequence. The one in b) has a narrow ridge that indicates a little overlapping region between parts that constitute this chimeric sequence. On the other hand, repetitive regions in pile-o-grams can be identified by large ridges as seen in pile-o-gram c).

At this point, each sequence is represented with a vector of unsigned short integers containing base coverages which is approximately equal to storing the complete FASTQ version of the input sequence file. Such pile-o-grams are first used to trim sequences to the largest region covered by at least three sequences, similarly to Miniasm [6]. Afterwards, Ra scans through the coverage vector to identify slopes in the signal by keeping sliding windows left and right of each position in the sequence. The maximal value of each window is compared to the current position and the slope is stored if the coverage ratio is large enough. Collected slopes are grouped into more complex shapes such as rifts and ridges, and each group is further investigated. To decrease the number of false positive annotations, we utilize the information about approximate sequencing depth. The pairwise overlaps are parsed again, contained sequences are dropped, internal overlaps are set aside, and only the remaining prefix-suffix overlaps are kept. They are used to group sequences in connected components with a simple depth-first search [14]. Coverage medians are calculated from pile-o-grams and a global median is obtained for each connected component separately. This way we can treat different sequencing depths correctly. The coverage median of a component is used to determine the relevance of each rift and ridge in all signals of that component. A rift is chimeric if it contains a base with coverage bellow the coverage median divided by 1.84, while ridges represent repetitive regions if majority of their bases have coverage above the coverage median multiplied with 1.42 (both values empirically determined). Narrow ridges located inside the middle of each sequence also undergo a chimeric test. During the second parsing of the overlap file they are declared chimeric if there are not at least three overlaps containing them. Once annotation is finished, Ra breaks sequences over rifts and chimeric ridges, retains the longest non-chimeric region and reconsiders corresponding overlaps that have been classified as internal beforehand. Remaining overlaps that either start or end inside a ridge located at a sequence end are removed, if at least one of overlapped sequences has another overlap that pierces through the ridge in question.

Ra was developed atop minimap2 because it is the fastest pairwise overlapper for long reads. Minimap2 and its predecessor minimap both filter out the most frequent k-mers in order to decrease the number of matching minimizers bound to be stored into memory and thus increases the execution speed [6] [12]. The affected k-mers mostly originate from repetitive regions or from higher copy-number molecules (e.g. plasmids). By employing the layout step hierarchically, first removing contained and chimeric sequences in the preconstruction step, we postpone the repeat annotation to the second step. The whole sequence set is mapped only against the leftover sequences (which constitute a tiny bit of the dataset) and the k-mer filter is decreased by at least one order of magnitude. Pile-o-grams are constructed anew and the increase in base coverage of repetitive regions greatly aids the annotation process.

As mentioned before, pile-o-grams can be treated as one-dimensional signals which makes them suitable for machine learning algorithms. Therefore, we also applied semi-supervised [15] and unsupervised learning algorithms [16] in order to aid sequence classification and annotation. Unfortunately, current results are still outperformed by heuristics.

B. Graph simplification

After preprocessing, the assembly graph is built and simplified stepwise. Transitive reduction is applied first, as described in [4]. Next, paths consisting of less than seven sequences and without any incoming edges are treated as dead ends

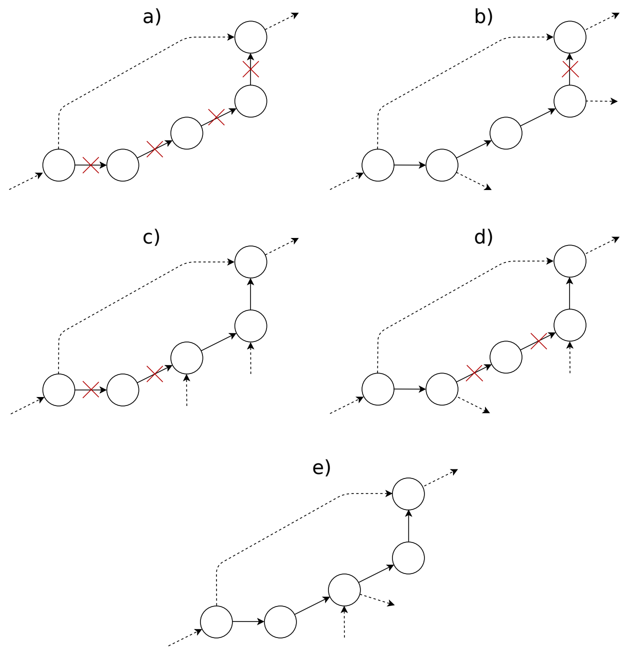

Fig. 3. Path types in bubble-like structures of the assembly graph. Paths in consideration consist of solid lines and are inspected for edges which removal will not discontinue any other path. Subfigure a) depicts the simplest scenario in which the whole path can be removed. When they are several nodes with multiple outgoing edges and a single incoming edge, everything after the last node can be deleted as in b). Similar rule applies to the case in c) where they are several nodes with multiple incoming edges and a single outgoing edge. Everything before the first such node can be deleted. When there is a combination of those node types, the edges after the last node with multiple outgoing edges and before the first node with multiple incoming edges can be deleted, as seen in d). This only applies when those two nodes do not have multiple edges of the other type, i.e. if a node has multiple incoming edges it may not have multiple outgoing edges (subfigure e)).

in the graph (tips) and are removed in an iterative fashion. Finally, bubble-like structures are found with breadth-first search similar to [17]. Each bubble is inspected for edges whose removal will not cause a break in any other path of the graph (Fig. 3). The path of the bubble with fewer sequences is examined first, and if such edges do not exist, the other path is considered. Bubbles are detected and popped iteratively as well.

Above described simplification methods coupled with sequence preprocessing are often enough to fully reconstruct the genome. Sequence preprocessing being a heuristic method can sometimes miss a portion of chimeric sequences and false overlaps, leaving us with tangles in the assembly graph. Miniasm solves its leftover tangles by removing short overlaps [6]. Ra follows the same approach but removes only those overlaps that are much shorter than any other overlap in a given tangle (4.2 times by default).

III. Results

We evaluated our de novo assembler on several publicly available Oxford Nanopore and Pacific Biosciences data sets. Results such as NG50, memory consumption and CPU time can be seen in Table II. We run Ra (commit *07364a1*) with 12 threads on a machine with Intel Xeon CPU (2.40GHz), and used QuastLG [18] (*v5.0.2*) and dnadiff [19] (*v1.3*) for evaluation.

We can observe how NG50 can still be improved and we believe that this is achievable with more relaxed constraints in sequence annotation and better heuristics for solving leftover tangles in the assembly graph. The memory consumption is bound by the sequence file plus some overhead for the overlap files which should be small enough for most use cases. The overlap and consensus steps dominate the execution time, but we believe it is possible to speed up the overlap step, which is the future path of optimizations we will consider.

Ra is a simple and lightweight assembler which happens to perform well on genomes sizes ranging from bacteria to plants shown by independent evaluations. For example, in a comparison of long read assembler applied to various bacterial datasets [20]), Ra was declared the most reliable assembler. In addition, it was used by [21] and yielded the most contiguous plant assemblies in their study.

References

[1] M. Jain, S. Koren, K. H. Miga, J. Quick, A. C. Rand, T. A. Sasani, J. R. Tyson, A. D. Beggs, A. T. Dilthey, I. T. Fiddes *et al.*, "Nanopore sequencing and assembly of a human genome with ultra-long reads," *Nature biotechnology*, vol. 36, no. 4, p. 338, 2018. doi: https://doi.org/10.1038/nbt.4060

[2] A. M. Wenger, P. Peluso, W. J. Rowell, P.-C. Chang, R. J. Hall, G. T. Concepcion, J. Ebler, A. Fungtammasan, A. Kolesnikov, N. D. Olson *et al.*, "Highly-accurate long-read sequencing improves variant detection and assembly of a human genome," *bioRxiv*, p. 519025, 2019. doi: https://doi.org/10.1101/519025

TABLE II
Assembly evaluation

Dataset	Species	Type	Size ($\times 10^6$)	Depth	Length	NG50	Identity	CPU time (min) / Memory (GB) Overlap step	Layout step	Consensus step
SRR5665597	*Klebsiella pneumoniae*	ONT	5.3	122	5453526	5344601	0.9841	26.39 / 8.32	3.71 / 3.19	116.45 / 2.22
ERR1140973	*Klebsiella pneumoniae*	PB	5.3	45	5548584	5299089	0.9915	4.20 / 3.45	1.10 / 1.02	22.04 / 0.79
SCONT*	*Saccharomyces cerevisiae*	ONT	12.1	59	12166217	288671	0.9740	36.01 / 8.62	4.49 / 3.03	73.53 / 2.20
SCPB**	*Saccharomyces cerevisiae*	PB	12.1	127	12267357	711780	0.9974	47.03 / 16.87	8.20 / 4.54	122.89 / 4.702
SRR6702603	*Drosophila melanogaster*	ONT	143.7	32	128449879	1854698	0.9884	652.06 / 22.94	53.35 / 10.82	570.28 / 13.75
SRX499318	*Drosophila melanogaster*	PB	143.7	109	135936009	2128249	0.9976	2302.29 / 34.15	676.85 / 51.92	1669.46 / 47.45

*S288C R9 dataset from http://www.genoscope.cns.fr/externe/Download/Projets/yeast/datasets/raw_data/

**W303 P4C2 dataset from https://github.com/PacificBiosciences/DevNet/wiki/Saccharomyces-cerevisiae-W303-Assembly-Contigs

[3] P. A. Pevzner, H. Tang, and M. S. Waterman, "An eulerian path approach to dna fragment assembly," *Proceedings of the national academy of sciences*, vol. 98, no. 17, pp. 9748–9753, 2001. doi: https://doi.org/10.1073/pnas.171285098

[4] E. W. Myers, "The fragment assembly string graph," *Bioinformatics*, vol. 21, no. suppl_2, pp. ii79–ii85, 2005. doi: https://doi.org/10.1093/bioinformatics/bti1114

[5] J. R. Miller, A. L. Delcher, S. Koren, E. Venter, B. P. Walenz, A. Brownley, J. Johnson, K. Li, C. Mobarry, and G. Sutton, "Aggressive assembly of pyrosequencing reads with mates," *Bioinformatics*, vol. 24, no. 24, pp. 2818–2824, 2008. doi: https://doi.org/10.1093/bioinformatics/btn548

[6] H. Li, "Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences," *Bioinformatics*, vol. 32, no. 14, pp. 2103–2110, 2016. doi: https://doi.org/10.1093/bioinformatics/btw152

[7] S. Koren, B. P. Walenz, K. Berlin, J. R. Miller, N. H. Bergman, and A. M. Phillippy, "Canu: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation," *Genome research*, vol. 27, no. 5, pp. 722–736, 2017. doi: https://doi.org/10.1101/gr.215087.116

[8] C.-S. Chin, P. Peluso, F. J. Sedlazeck, M. Nattestad, G. T. Concepcion, A. Clum, C. Dunn, R. O'Malley, R. Figueroa-Balderas, A. Morales-Cruz *et al.*, "Phased diploid genome assembly with single-molecule real-time sequencing," *Nature methods*, vol. 13, no. 12, p. 1050, 2016. doi: https://doi.org/10.1038/nmeth.4035

[9] M. Kolmogorov, J. Yuan, Y. Lin, and P. A. Pevzner, "Assembly of long, error-prone reads using repeat graphs," *Nature biotechnology*, vol. 37, no. 5, p. 540, 2019. doi: https://doi.org/10.1038/s41587-019-0072-8

[10] G. M. Kamath, I. Shomorony, F. Xia, T. A. Courtade, and N. T. David, "Hinge: long-read assembly achieves optimal repeat resolution," *Genome research*, vol. 27, no. 5, pp. 747–756, 2017. doi: https://doi.org/10.1101/gr.216465.116

[11] J. Ruan and H. Li, "Fast and accurate long-read assembly with wtdbg2," *BioRxiv*, p. 530972, 2019. doi: https://doi.org/10.1101/530972

[12] H. Li, "Minimap2: pairwise alignment for nucleotide sequences," *Bioinformatics*, vol. 34, no. 18, pp. 3094–3100, 2018. doi: https://doi.org/10.1093/bioinformatics/bty191

[13] R. Vaser, I. Sović, N. Nagarajan, and M. Šikić, "Fast and accurate de novo genome assembly from long uncorrected reads," *Genome research*, vol. 27, no. 5, pp. 737–746, 2017. doi: https://doi.org/10.1101/gr.214270.116

[14] J. Hopcroft and R. Tarjan, "Algorithm 447: efficient algorithms for graph manipulation," *Communications of the ACM*, vol. 16, no. 6, pp. 372–378, 1973. doi: https://doi.org/10.1145/362248.362272

[15] T. Šebrek, J. Tomljanović, J. Krapac, and M. Šikić, "Read classification using semi-supervised deep learning," *arXiv preprint arXiv:1904.10353*, 2019.

[16] J. Tomljanović, T. Šebrek, and M. Šikić, "Unsupervised learning of sequencing read types," in *Proceedings of the 2017 International Conference on Computational Biology and Bioinformatics*. ACM, 2017. doi: https://doi.org/10.1145/3155077.3155080 pp. 12–17.

[17] D. R. Zerbino and E. Birney, "Velvet: algorithms for de novo short read assembly using de bruijn graphs," *Genome research*, vol. 18, no. 5, pp. 821–829, 2008. doi: https://doi.org/10.1101/gr.074492.107

[18] A. Mikheenko, A. Prjibelski, V. Saveliev, D. Antipov, and A. Gurevich, "Versatile genome assembly evaluation with quast-lg," *Bioinformatics*, vol. 34, no. 13, pp. i142–i150, 2018. doi: https://doi.org/10.1093/bioinformatics/bty266

[19] A. L. Delcher, S. L. Salzberg, and A. M. Phillippy, "Using mummer to identify similar regions in large sequence sets," *Current protocols in bioinformatics*, no. 1, pp. 10–3, 2003. doi: https://doi.org/10.1002/0471250953.bi1003s00

[20] R. Wick, "rrwick/Long-read-assembler-comparison: Initial release," May 2019. doi: https://doi.org/10.5281/zenodo.2702443

[21] C. Belser, B. Istace, E. Denis, M. Dubarry, F.-C. Baurens, C. Falentin, M. Genete, W. Berrabah, A.-M. Chèvre, R. Delourme *et al.*, "Chromosome-scale assemblies of plant genomes using nanopore long reads and optical maps," *Nature plants*, vol. 4, no. 11, p. 879, 2018. doi: https://doi.org/10.1038/s41477-018-0289-4