Fast and accurate de novo genome assembly from long uncorrected reads

Robert Vaser,[1,5] Ivan Sović,[2,5] Niranjan Nagarajan,[3] and Mile Šikić[1,4]

[1] *Department of Electronic Systems and Information Processing, University of Zagreb, Faculty of Electrical Engineering and Computing, 10000 Zagreb, Croatia;* [2] *Centre for Informatics and Computing, Ruđer Bošković Institute, 10000 Zagreb, Croatia;* [3] *Genome Institute of Singapore, Singapore 138672, Singapore;* [4] *Bioinformatics Institute, Singapore 138671, Singapore*

The assembly of long reads from Pacific Biosciences and Oxford Nanopore Technologies typically requires resource-intensive error-correction and consensus-generation steps to obtain high-quality assemblies. We show that the error-correction step can be omitted and that high-quality consensus sequences can be generated efficiently with a SIMD-accelerated, partial-order alignment–based, stand-alone consensus module called Racon. Based on tests with PacBio and Oxford Nanopore data sets, we show that Racon coupled with miniasm enables consensus genomes with similar or better quality than state-of-the-art methods while being an order of magnitude faster.

[Supplemental material is available for this article.]

With the advent of long-read sequencing technologies from Pacific Biosciences (PacBio) and Oxford Nanopore Technologies (ONT), the ability to produce genome assemblies with high contiguity has received a significant fillip. However, to cope with the relatively high error rates (>5%) of these technologies, assembly pipelines have typically relied on resource-intensive error-correction (of reads) and consensus-generation (from the assembly) steps (Chin et al. 2013; Loman et al. 2015). More recent methods such as FALCON (Chin et al. 2016) and Canu (Koren et al. 2017) have refined this approach and have significantly improved runtimes but are still computationally demanding for large genomes (Sović et al. 2016a). Recently, Li (2016) showed that long erroneous reads can be assembled without the need for a time-consuming error-correction step. The resulting assembler, miniasm, is an order of magnitude faster than other long-read assemblers but produces sequences that can have more than 10 times as many errors as other methods (Sović et al. 2016a). As fast and accurate long-read assemblers can enable a range of applications, from more routine assembly of mammalian and plant genomes to structural variation detection, improved metagenomic classification, and even online, "read until" assembly (Loose et al. 2016), a fast and accurate consensus module is a critical need. This was also noted by Li (2016), highlighting that fast assembly was only feasible if a consensus module matching the speed of minimap and miniasm was developed as well.

Here we address this need by providing a very fast consensus module called Racon (for Rapid Consensus), which when paired with a fast assembler such as miniasm, enables the efficient construction of genome sequences with high accuracy even without an error-correction step. Racon provides a stand-alone, platform-independent consensus module for long and erroneous reads and can also be used as a fast and accurate read correction tool.

Results

Racon is designed as a user-friendly stand-alone consensus module that is not explicitly tied to any de novo assembly method or sequencing technology. It reads multiple input formats (GFA, FASTA, FASTQ, SAM, MHAP, and PAF), allowing simple interoperability and modular design of new pipelines. Even though other stand-alone consensus modules, such as Quiver (Chin et al. 2013) and Nanopolish (Loman et al. 2015) exist, they require sequencer-specific input and are intended to be applied *after* the consensus phase of assembly to further polish the sequence. Racon is run with sequencer-independent input (only uses base quality values), is robust enough to work with uncorrected read data, and is designed to rapidly generate high-quality consensus sequences. These sequences can be further polished with Quiver or Nanopolish or by applying Racon for more iterations.

Racon takes as input a set of raw backbone sequences, a set of reads, and a set of mappings between reads and backbone sequences. Mappings can be generated using any mapper/overlapper that supports the MHAP, PAF, or SAM output formats, such as minimap (Li 2016), MHAP (Berlin et al. 2015), or GraphMap (Sović et al. 2016b). In our tests, we used minimap as the mapper as it was the fastest and provided reasonable results. Racon uses the mapping information to construct a partial-order alignment (POA) graph, using a single instruction multiple data (SIMD) implementation to accelerate the process (SPOA). More details on Racon and SPOA can be found in the Methods section.

For the purpose of evaluation, we paired Racon with minimap (as an overlapper) and miniasm to form a fast and accurate de novo assembly pipeline (referred to here as *miniasm+Racon* for short). Our complete pipeline is thus composed of four steps: (1) minimap for overlap detection, (2) miniasm layout for generating raw contigs, (3) minimap for mapping of raw reads to raw contigs, and (4) Racon for generating high-quality consensus sequences. Steps three and four can be run multiple times to achieve further iterations of the consensus. We then compared the miniasm+Racon

[5]These authors contributed equally to this work.
Corresponding author: mile.sikic@fer.hr
Article published online before print. Article, supplemental material, and publication date are at http://www.genome.org/cgi/doi/10.1101/gr.214270.116.

pipeline to other state-of-the-art de novo assembly tools for third-generation sequencing data (i.e., FALCON and Canu). Note that FALCON and Canu have previously been benchmarked with other assembly methods such as PBcR and a pipeline from Loman et al. (2015) and shown to produce high-quality assemblies with improved running times (Sović et al. 2016a). Assembly pipelines were evaluated in terms of consensus sequence quality (Table 1), runtime and memory usage (Table 2; Fig. 1), and scalability with respect to genome size (Fig. 2) on several PacBio and Oxford Nanopore data sets (see Methods).

As can be seen from Table 1, all assembly pipelines were able to produce assemblies with high coverage of the reference genome and in a few contigs. Canu, FALCON, and the miniasm+Racon pipeline also constructed sequences with comparable sequence identity to the reference genome, with the iterative use of Racon serving as a polishing step for obtaining higher sequence identity. In addition, the miniasm+Racon pipeline was found to be significantly faster for all data sets, with a 5× to 301× speedup compared with Canu and 9× to 218× speedup compared with FALCON (Fig. 1).

Racon's speedup was more pronounced for larger genomes and is likely explained by the observation that it scales linearly with genome size (for fixed coverage) (Fig. 2).

The runtime of the miniasm+Racon pipeline was dominated by the time for the consensus-generation step in Racon, highlighting that this step is still the most compute intensive one for small genomes (Table 3). However, the results in Table 3 suggest that for

Table 1. Assembly and consensus results across six data sets of varying genome length and sequencing data type

	miniasm+Racon: one iteration	miniasm+Racon: two iterations	Canu	FALCON
Lambda ONT R7.3 30×				
Ref. genome size (in bp)	48,502	48,502	48,502	48,502
Total bases (in bp)	47,916	47,872	25,077	7212
No. of ref. chromosomes	1	1	1	1
No. of contigs	1	1	1	1
Aln. bases ref. (in bp)	48,438 (99.87%)	48,434 (99.86%)	25,833 (53.26%)	7483 (15.43%)
Aln. bases query (in bp)	47,916 (100.00%)	47,872 (100.00%)	25,077 (100.00%)	7212 (100.00%)
Avg. identity	97.59	97.97	96.87	95.77
Escherichia coli K-12 ONT R7.3 54×				
Ref. genome size (in bp)	4,641,652	4,641,652	4,641,652	4,641,652
Total bases (in bp)	4,637,173	4,632,058	4,601,503	4,580,230
No. of ref. chromosomes	1	1	1	1
No. of contigs	1	1	1	1
Aln. bases ref. (in bp)	4,640,867 (99.98%)	4,641,323 (99.99%)	4,631,173 (99.77%)	4,627,613 (99.70%)
Aln. bases query (in bp)	4,636,689 (99.99%)	4,632,055 (100.00%)	4,601,365 (100.00%)	4,580,230 (100.00%)
Avg. identity	99.13	99.32	99.28	98.84
Saccharomyces cerevisiae S288C ONT R9 59×				
Ref. genome size (in bp)	12,157,105	12,157,105	12,157,105	12,157,105
Total bases (in bp)	12,172,019	12,167,797	12,224,535	11,643,917
No. of ref. chromosomes	17	17	17	17
No. of contigs	42	42	45	41
Aln. bases ref. (in bp)	12,104,541 (99.57%)	12,110,095 (99.61%)	12,120,070 (99.70%)	11,885,904 (97.77%)
Aln. bases query (in bp)	12,108,082 (99.48%)	12,115,796 (99.57%)	12,196,684 (99.77%)	11,643,482 (100.00%)
Avg. identity	97.88	98.04	98.61	98.22
E. coli K-12 PacBio P6C4 160×				
Ref. genome size (in bp)	4,641,652	4,641,652	4,641,652	4,641,652
Total bases (in bp)	4,653,199	4,645,508	4,664,416	4,666,788
No. of ref. chromosomes	1	1	1	1
No. of contigs	1	1	1	1
Aln. bases ref. (in bp)	4,641,501 (100.00%)	4,641,439 (100.00%)	4,641,652 (100.00%)	4,641,652 (100.00%)
Aln. bases query (in bp)	4,653,111 (100.00%)	4,645,508 (100.00%)	4,664,416 (100.00%)	4,666,788 (100.00%)
Avg. identity	99.63	99.90	99.99	99.90
S. cerevisiae W303 PacBio P4C2 127×				
Ref. genome size (in bp)	12,157,105	12,157,105	12,157,105	12,157,105
Total bases (in bp)	12,071,278	12,051,573	12,402,332	12,003,077
No. of ref. chromosomes	17	17	17	17
No. of contigs	30	30	29	44
Aln. bases ref. (in bp)	12,023,607 (98.90%)	12,025,677 (98.92%)	12,127,627 (99.76%)	11,932,488 (98.15%)
Aln. bases query (in bp)	12,046,299 (99.79%)	12,027,338 (99.80%)	12,363,941 (99.69%)	11,910,549 (99.23%)
Avg. identity	99.43	99.72	99.86	99.70
Caenorhabditis elegans PacBio P6C4 81×				
Ref. genome size (in bp)	100,272,607	100,272,607	100,272,607	100,272,607
Total bases (in bp)	106,353,704	106,392,402	106,687,886	105,858,394
No. of ref. chromosomes	6	6	6	6
No. of contigs	77	77	134	242
Aln. bases ref. (in bp)	100,017,898 (99.75%)	99,979,140 (99.71%)	100,166,301 (99.89%)	99,295,695 (99.03%)
Aln. bases query (in bp)	101,711,974 (95.64%)	101,741,297 (95.63%)	102,928,910 (96.48%)	102,008,289 (96.36%)
Avg. identity	99.44	99.73	99.89	99.74

Table 2. Time/memory consumption for all data sets

	miniasm+Racon: one iteration	miniasm+Racon: two iterations	Canu	FALCON
Lambda ONT 30×	0.12/<0.1	0.25/<0.1	2.86/1.90	2.29/0.854
E. coli K-12 ONT R7.3 54×	25/3.32	46/3.32	1328/4.03	829/12.29
S. cerevisiae S288C ONT R9 59×	28/7.84	44/7.84	13,243/4.88	9603/8.82
E. coli K-12 PacBio P6C4 160×	86/9.69	162/9.69	773/3.60	2908/9.93
S. cerevisiae W303 PacBio P4C2 127×	115/16.09	215/16.09	6375/3.65	14,808/4.78
C. elegans PacBio P6C4 81×	1247/50.38	2004/50.38	37,853/10.16	119,766/7.59

Results are presented in the following format: CPU time (in min)/Maximum memory (in GB).

larger genomes the mapping computation stage can catch up in terms of resource usage. Furthermore, if a polishing stage is used, this would typically be more resource intensive.

Comparison of the results of the various assembly pipelines after a polishing stage confirmed that the use of Racon provided better results than just the miniasm assembly (average identity of 99.80% vs. 98.06%) and that the miniasm+Racon assembly matched the best reported sequence quality for this data set (from the Loman et al. pipeline) (Sović et al. 2016a), while providing a better match to the *actual size* of the reference genome (4,641,652 bp) (Table 4). We additionally observed that Nanopolish executed more than 6× faster on miniasm+Racon contigs than on raw miniasm assemblies (248.28 CPUh vs. 1561.80 CPUh), and the miniasm+Racon+Nanopolish approach achieved the same sequence quality as the original Loman et al. pipeline, while being much faster.

We also evaluated Racon's use as an error-correction module. The three main conceptual differences between error-correction and consensus modes in Racon are that (1) instead of mapping reads to a reference, overlapping of reads needs to be performed (e.g., using minimap with overlap parameters instead of defaults); (2) multiple overlaps are not filtered like mappings (when mapping, we only want to find one best mapping position); and (3) the parallelization is not conducted on a per-window level, but on a per-read level to reduce the system time; i.e., each read consumes one thread. We noted that Racon–corrected reads had error rates comparable to FALCON and Canu but provided better coverage of the genome (Table 5). Overall, Nanocorrect (Loman et al. 2015) had the best results in terms of error rate, but it had lower reference coverage and was more than two orders of magnitude slower than Racon.

Finally, the default mode of Racon determines the consensus by splicing results of nonoverlapping windows together. We additionally attempted to assess the influence of using overlapping windows, where we extended each window by 10% on both of its ends. Neighboring windows were then aligned and clipped to form the final consensus sequence. By using this approach, we re-evaluated the consensus quality (Supplemental Table S1), the speed (Supplemental Table S2), and the quality of error correction (Supplemental Table S3) to the default Racon mode on all data sets. When using the window overlapping mode, the consensus quality increased (up to 0.06%) in all but one case. The most significant increase was obtained on the *E. coli* PacBio data set, where average identity rose from 99.90% to 99.94% This improvement came at the expense of higher running time (≈10%–15%) and an additional heuristic (window overlap length), which is why we enabled this mode as a nondefault option within the Racon module.

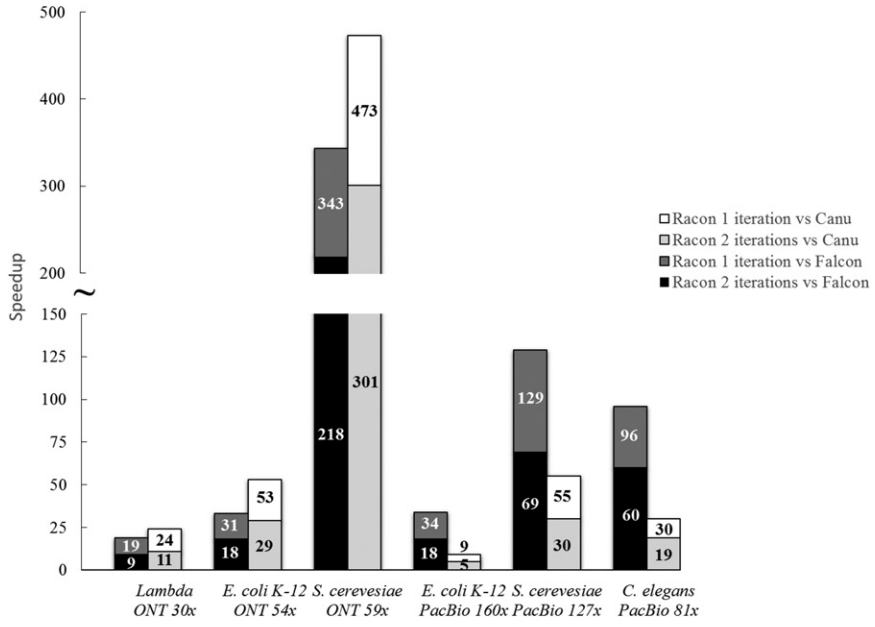

Figure 1. Racon's speedup compared with FALCON and Canu.

Recently, another stand-alone consensus module called Sparc (Ye and Ma 2016) emerged. We evaluated Sparc in a similar manner to Racon, with the only difference being in the mapping step (step 3 in the miniasm+Racon pipline) where Sparc uses BLASR (Chaisson and Tesler 2012). We named this pipeline miniasm+Sparc accordingly. The results of the comparison are shown in Table 6, which contains consensus sequence quality, runtime, and memory usage. It can be seen that the consensus quality of the miniasm+Sparc pipeline is consistently lower than that of the miniasm +Racon pipeline. In addition, miniasm +Sparc is up to 10× slower on larger genomes. Also, we noticed that Sparc crashed on some contigs due to unknown reasons.

Discussion

The principal contribution of this work is to take the concept of fast, error-correction-free, long-read assembly, as

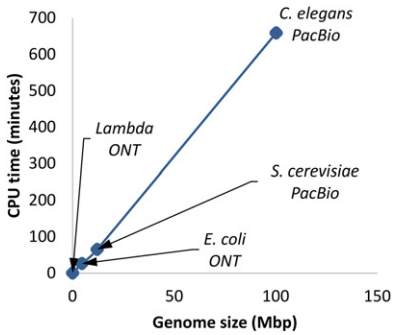

Figure 2. Scalability of Racon as a function of genome size. Read coverage was subsampled to be 81× (limited by the *Caenorhabditis elegans* data set), and the figure shows results for one iteration of Racon.

embodied by the recently developed program miniasm, to its logical end. Miniasm is remarkably efficient and effective in taking erroneous long reads and producing contig sequences that are structurally accurate (Sović et al. 2016a). However, assemblies from miniasm do not match up in terms of sequence quality compared with the best assemblies that can be produced with existing assemblers. This serves as a significant barrier for adopting this "light-weight" approach to assembly, despite its attractiveness for greater adoption of de novo assembly methods in genomics. In this work, we show that the sequence quality of a correction-free assembler can indeed be efficiently boosted to a quality comparable to other resource-intensive, state-of-the-art assemblers. This makes the tradeoff offered much more attractive, and the concept of a correction-free assembler more practically useful. Racon is able to start from uncorrected contigs and raw reads and still generate accurate

Table 3. Resource usage for various parts of the miniasm+Racon assembly pipeline

	Lambda ONT	*E. coli* ONT	*S. cerevisiae* ONT	*E. coli* PacBio	*S. cerevisiae* PacBio	*C. elegans* PacBio
minimap overlap	<0.1/<0.1	2.70/3.32	10.21/7.84	11.23/9.69	21.90/16.09	473.82/50.38
miniasm	<0.1/<0.1	<0.1/<0.1	0.24/0.22	0.46/0.40	0.56/0.46	4.22/3.45
minimap mapping: first iteration	<0.1/<0.1	0.24/0.24	0.83/0.27	0.65/0.23	1.48/0.26	13.82/1.06
Racon consensus: first iteration	0.11/<0.1	22.16/2.91	17.06/2.11	73.21/7.88	91.55/4.69	755.29/22.11
minimap mapping: second iteration	<0.1/<0.1	0.27/0.23	0.89/0.30	0.72/0.24	1.63/0.27	15.15/0.96
Racon consensus: second iteration	0.13/<0.1	20.82/2.25	14.85/1.98	75.75/6.40	97.94/4.61	741.68/20.71
Total CPU time/maximum memory	0.25/<0.1	46/3.32	44/7.84	162/9.69	215/16.09	2004/50.38

Results are presented in the following format: CPU time (in min)/Maximum memory (in GB).

Table 4. Results after polishing assemblies with Nanopolish on the *E. coli* K-12 ONT R7.3 54× data set

	Raw miniasm	miniasm+Racon: two iterations	Canu	FALCON	Loman et al. pipeline
Total bases (in bp)	4,696,482	4,641,756	4,631,443	4,624,811	4,695,512
|Total bases−Genome size| (in bp)	54,830	104	10,209	16,841	53,860
Aligned bases ref. (in bp)	4,635,941 (99.88%)	4,641,312 (99.99%)	4,633,324 (99.82%)	4,627,571 (99.70%)	4,641,325 (99.99%)
Aligned bases query (in bp)	4,687,686 (99.81%)	4,641,756 (100.00%)	4,631,361 (100.00%)	4,624,811 (100.00%)	4,695,463 (100.00%)
Average identity	98.06	99.80	99.80	99.78	99.80

Table 5. Comparison of error-correction modules on *E. coli* K-12 ONTR7.3 54× data set

	CPU time (in h)	Coverage	Insertion rate (%)	Deletion rate (%)	Mismatch rate (%)	Match rate (%)	Error rate (I+D+M) (%)
Raw	—	53.55×	5.23	2.83	4.89	89.81	12.95
Racon	10	50.23×	0.52	0.58	0.23	99.25	1.33
Nanocorrect	8100	44.74×	0.14	0.43	0.03	99.83	0.60
FALCON	22	46.95×	0.04	1.11	0.06	99.90	1.21
Canu	5	35.53×	0.06	1.25	0.08	99.85	1.39

Values presented in the table are median values of the error and match rate estimates.

Table 6. Quality, time, and memory comparison of Racon and Sparc as consensus modules in a de novo assembly pipeline composed of minimap as an overlapper and miniasm as a layout module

	miniasm+Racon: one iteration	miniasm+Racon: two iterations	miniasm+Sparc: one iteration	miniasm+Sparc: two iterations
Lambda ONT R7.3 30×				
Average identity	97.59	97.97	95.04	96.87
CPU time (in min)	0.12	0.25	<0.1	0.14
Maximum memory (in GB)	<0.1	<0.1	<0.1	<0.1
E. coli K-12 ONT R7.3 54×				
Average identity	99.13	99.32	98.10	99.16
CPU time (in min)	25	46	28	54
Maximum memory (in GB)	3.32	3.32	3.37	3.37
S. cerevisiae S288C ONT R9 59×				
Average identity	97.88	98.04	97.77	98.02
CPU time (in min)	28	44	210	424
Maximum memory (in GB)	7.84	7.84	7.84	7.84
E. coli K-12 PacBio P6C4 160×				
Average identity	99.63	99.90	99.34	99.76
CPU time (in min)	86	162	102	187
Maximum memory (in GB)	9.69	9.69	9.72	9.72
S. cerevisiae W303 PacBio P4C2 127×				
Average identity	99.43	99.72	99.09	99.57
CPU time (in min)	115	215	297	587
Maximum memory (in GB)	16.09	16.09	16.09	16.09
C. elegans PacBio P6C4 81×				
Average identity	99.44	99.73	99.16	99.57
CPU time (in min)	1247	2004	4032	8915
Maximum memory (in GB)	50.38	50.38	51.80	51.80

Both consensus tools were run on identical sets of raw contigs.

sequences efficiently because it exploits the development of a SIMD version of the robust POA framework. This makes the approach scalable to large genomes and general enough to work with data from very different sequencing technologies. With the increasing interest in the development of better third-generation assembly pipelines, we believe that Racon can serve as useful plug-in consensus module that enables software reuse and modular design.

We would also like to note that while miniasm is a very fast tool that produces correct genome structures, its current implementation is not optimized for long repetitive genomes (Li 2016). Miniasm loads all overlaps into RAM, which might cause crashes due to insufficient memory resources. Although the focus of this study is on the miniasm+Racon pipeline, Racon is designed as a general purpose consesus module that can be coupled with other assembly pipelines as a consensus or polishing step.

Methods

Racon is based on the *Partial Order Alignment* (POA) graph approach (Lee et al. 2002; Lee 2003), and we report the development of a SIMD version that significantly accelerates this analysis. An overview of Racon's steps is given in Figure 3. The entire process is also shown in detail in Algorithm 1.

To perform consensus calling (or error-correction), Racon depends on an input set of query-to-target mappings (query is the set of reads, while a target is either a set of contigs in the consensus context or a set of reads in the error-correction context) as well as quality values of the input reads. Racon then loads the mappings and performs simple filtering (Algorithm 1, lines 1–3; Algorithm 2): (1) at most one mapping per read is kept in consensus context (in error-correction context this particular filtering is disabled), and (2) mappings that have a high error rate (i.e. $|1 - \min(d_q,d_t)/\max(d_q,d_t)| \geq e$, where d_q and d_t are the lengths of the mapping in the query and the target, respectively, and e is a user-specified error-rate threshold) are removed. For each mapping that survived the filtering process, a fast edit-distance–based alignment is performed (Algorithm 1, lines 4–10; Myers 1999). We used the Edlib implementation of the Myers algorithm (Šošić and Šikić 2016). This alignment is needed only to split the reads into chunks that fall into particular nonoverlapping windows on the backbone sequence. Chunks of reads with an average quality lower than a

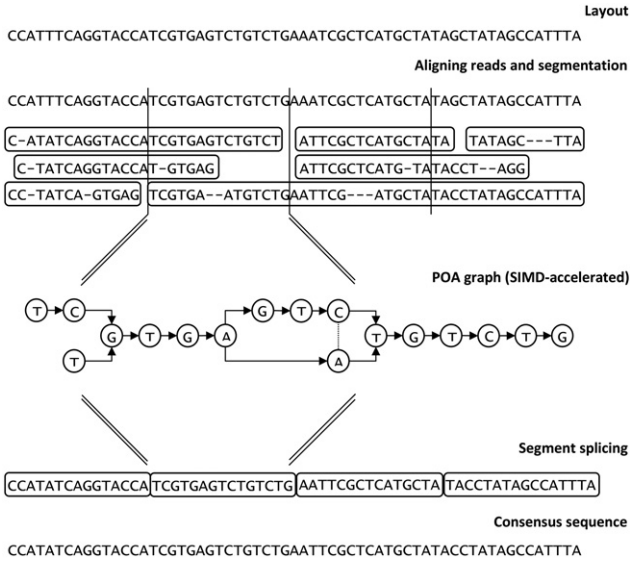

Figure 3. Overview of the Racon consensus process.

Input: Set of noisy target sequences (e.g. contigs) $\mathcal{T}$ to perform consensus on, set of read sequences $\mathcal{Q}$ where each $q \in \mathcal{Q}$ is a tuple consisting of $(seq, qual)$ (sequence data and quality values), set of mappings (or overlaps) $\mathcal{O}$ between $\mathcal{Q}$ and $\mathcal{T}$, window size w, minimum allowed quality value qv_{min}, mode m to run the Racon in (can be either "*consensus*" or "*errorcorrection*"), and maximum expected error rate e. A mapping/overlap $o \in \mathcal{O}$ is a tuple of 12 values: $(q_{id}, t_{id}, p_{cb}, n_k, q_{rev}, q_s, q_e, q_{len}, t_{rev}, t_s, t_e, t_{len})$, where q_{id} and t_{id} are the query and target sequence IDs, p_{cb} percentage of query bases covered by the overlap, n_k number of exact matching seeds shared between the query and the target used to construct the mapping/overlap, q_{rev} and t_{rev} are 0 if query and target are forward oriented and 1 otherwise, q_s, q_e, t_s, t_e are the start and end coordinates of the mapping/overlap in the query and target, and q_{len}, t_{len} total query and target sequence lengths.

Output: Set of corrected target sequences $\mathcal{T}_c$

Function RACON($\mathcal{T}, \mathcal{Q}, \mathcal{O}, w, qv_{min}, m, e$) **begin**

$\mathcal{O}_f \leftarrow FilterErroneousMappings(\mathcal{O}, e)$
if $m \neq$ "*errorcorrection*" **then**
  $\mathcal{O}_f \leftarrow FilterUniqueMappings(\mathcal{O}_f)$
$\mathcal{A} \leftarrow$ empty hash table of sets ▷ *Aligned mappings/overlaps grouped by* t_{id}
foreach $o \in \mathcal{O}_f$ **do** ▷ *Align filtered mappings/overlaps*
  $q_{seq} \leftarrow \mathcal{Q}[o.q_{id}].seq[o.q_s ... o.q_e]$
  $q_{qual} \leftarrow \mathcal{Q}[o.q_{id}].qual[o.q_s ... o.q_e]$
  $t_{seq} \leftarrow \mathcal{T}[o.t_{id}][o.t_s ... o.t_e]$
  $aln \leftarrow$ bit-vector global alignment between q_{seq} and t_{seq}
  $\mathcal{A}[o.t_{id}] \leftarrow \mathcal{A}[o.t_{id}] \cup (o, aln, q_{seq}, q_{qual}, t_{seq})$
$\mathcal{T}_c \leftarrow \emptyset$ ▷ *A set for consensus sequences*
foreach $t_{id} \in \mathcal{A}$ **do** ▷ *Process each target sequence individually*
  $\mathcal{A}_t \leftarrow \mathcal{A}[t_{id}]$
  $t_c \leftarrow$ empty string
  for $w_{id} = 0$ *to* $\lceil \mathcal{T}[t_{id}]/w \rceil$ *step* w **do** ▷ *Process a target in non-overlapping windows*
    $w_s \leftarrow w_{id} \cdot w$ ▷ *Window start position on target seq*
    $w_e \leftarrow (w_{id} + 1) \cdot w - 1$ ▷ *Window end position on target seq*
    $s_{tseq} \leftarrow \mathcal{T}[t_{id}][w_s ... w_e]$ ▷ *Noisy target subsequence for SPOA*
    $s_{tqual} \leftarrow$ an array of $|s_{tseq}|$ elements initialized to 0 ▷ *Dummy QVs*
    $\mathcal{S} \leftarrow \emptyset$
    foreach $(o, aln, q_{seq}, q_{qual}, t_{seq}) \in \mathcal{A}_t$ **do** ▷ *Get subsequences of all reads*
      $(s_{qseq}, s_{qqual}) \leftarrow$ extract a substring of s_{qseq} which aligns between w_s and w_e on target with corresponding quality values q_{qqual}
      if $average(s_{qqual}) \geq qv_{min}$ **then**
        $\mathcal{S} \leftarrow \mathcal{S} \cup (s_{qseq}, s_{qqual})$
    $\mathcal{G} \leftarrow BuildGraph(s_{tseq}, s_{tqual})$ ▷ *As described in (Lee et al. 2002)*
    foreach $(s_{qseq}, s_{qqual}) \in \mathcal{S}$ **do**
      $\mathcal{G} \leftarrow AlignSequenceToGraphNW(\mathcal{G}, s_{qseq}, s_{qqual}, 5, -4, -8, -6)$
    $cons \leftarrow GenerateConsensus(\mathcal{G})$ ▷ *As described in (Lee 2003)*
    $t_c \leftarrow$ concatenate $cons$ to t_c
  $\mathcal{T}_c \leftarrow \mathcal{T}_c \cup t_c$
return $\mathcal{T}_c$

Algorithm 1. The Racon algorithm for consensus generation.

predefined threshold are removed from the corresponding window. With this quality filter, we are able to use only high-quality parts of reads in the final consensus. Each window is then processed independently in a separate thread by constructing a POA graph using SIMD acceleration and calling the consensus of the window. Quality values are used again during POA graph construction, where each edge is weighted by the sum of qualities of its source and destination nodes (bases; logarithmic domain). The final consensus sequence is then constructed by splicing the individual window consensuses together (per contig or read to be corrected).

POA and SIMD vectorization

POA performs multiple sequence alignment (MSA) through a directed acyclic graph (DAG), where nodes are individual bases of input sequences, and weighted, directed edges represent whether two bases are neighboring in any of the sequences. Weights of the edges represent the multiplicity (coverage) of each transition. Alternatively, weights can be set according to the base qualities of sequenced data. The alignment is carried out directly through dynamic programming (DP) between a new sequence and a prebuilt graph. While the regular DP for pairwise alignment has

Input: A set of mappings (or overlaps) $\mathcal{O}$ and expected maximum error rate e. Mappings/overlaps are defined the same as in Algorithm 1.
Output: Set of filtered mappings/overlaps $\mathcal{O}_f$

Function FILTERERRONEOUSMAPPINGS($\mathcal{O}, e$) **begin**

1 $\mathcal{O}_f \leftarrow \emptyset$
2 **foreach** $o \in \mathcal{O}$ **do**
3 $o_{max} \leftarrow \max(|o.q_e - o.q_s|, |o.t_e - o.t_s|)$ ▷ *Maximum mapping/overlap length*
4 $o_{min} \leftarrow \min(|o.q_e - o.q_s|, |o.t_e - o.t_s|)$ ▷ *Minimum mapping/overlap length*
5 **if** $(1 - o_{min}/o_{max}) > e$ **then** ▷ *Max mapping/overlap indel error rate*
6 **continue**
7 $\mathcal{O}_f \leftarrow \mathcal{O}_f \cup o$
8 **return** $\mathcal{O}_f$

Input: A set of mappings (or overlaps) $\mathcal{O}$. Mappings/overlaps are defined the same as in Algorithm 1.
Output: Set of filtered mappings/overlaps $\mathcal{O}_f$

Function FILTERUNIQUEMAPPINGS($\mathcal{O}$) **begin**

9 $\mathcal{H} \leftarrow$ empty hash table
10 **foreach** $o \in \mathcal{O}$ **do**
11 **if** $o.q_{id} \in \mathcal{H}$ **then** ▷ *Keep only one mapping/overlap per query (read)*
12 **if** $o.n_k > \mathcal{H}[o.q_{id}].n_k$ **then**
13 $\mathcal{H}[o.q_{id}] \leftarrow o$
14 **else**
15 $\mathcal{H}[o.q_{id}] \leftarrow o$
16 $\mathcal{O}_f \leftarrow \emptyset$
17 **foreach** $q_{id} \in \mathcal{H}$ **do**
18 $\mathcal{O}_f \leftarrow \mathcal{O}_f \cup \mathcal{H}[q_{id}]$
19 **return** $\mathcal{O}_f$

Algorithm 2. Functions for filtering mappings/overlaps in Racon.

time complexity of $O(3nm)$, where n and m are the lengths of the sequences being aligned, the sequence to graph alignment has a complexity of $O((2n_p+1)n|V|)$, where n_p is the average number of predecessors in the graph and $|V|$ is the number of nodes in the graph (Lee et al. 2002).

Consensus sequences are obtained from a built POA graph by performing a topological sort and processing the nodes from left to right. For each node v, the highest-weighted in-edge e of weight e_w is chosen, and a score is assigned to v such that scores[v] = e_w + scores[w], where w is the source node of the edge e (Lee 2003). The node w is marked as a predecessor of v, and a final consensus is generated by performing a traceback from the highest-scoring node r. In case r is an internal node (r has out edges), Lee (2003) proposed the idea of branch completion, where all scores for all nodes except scores[r] would be set to a negative value, and the traversal would continue from r as before, with the only exception that nodes with negative scores could not be added as predecessors to any other node.

One important advantage of POA is its speed, with its linear time complexity in the number of sequences (Lee et al. 2002). However, implementations of POA in current error-correction modules, such as Nanocorrect, are prohibitively slow for larger data sets. In order to increase the speed of POA while retaining its robustness, we explored a SIMD version of the algorithm (SPOA).

SPOA (Fig. 4; Algorithm 3) is inspired by the Rognes and Seeberg Smith-Waterman intra-set parallelization approach (Rognes and Seeberg 2000). It places the SIMD vectors parallel to the query sequence (the read), while placing a graph on the other dimension of the DP matrix (Fig. 4). In our implementation, the matrices used for tracking the maximum local-alignment scores ending in gaps are stored entirely in memory (Algorithm 3, lines 8 and 10). These matrices are needed to access scores of predecessors of particular nodes during alignment. Unlike a regular Gotoh alignment (Gotoh 1982), for each row in the POA DP matrix, all its predecessors (via in-edges of the corresponding node in graph) need to be processed as well (Algorithm 3, line 17). All columns are then processed using SIMD operations in a query-parallel manner, and the values of Gotoh's vertical matrix (Algorithm 3, line 20) and a partial update to Gotoh's main scoring matrix (Algorithm 3, line 24) are calculated. SIMD operations in Algorithm 3 process eight cells of the DP matrix at a time (16-bit registers). A temporary variable is used to keep the last cell of the

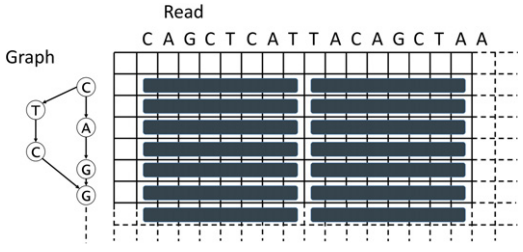

Figure 4. Depiction of the SIMD vectorization approach used in SPOA.

Input: A POA graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is a set of vertices and $\mathcal{E}$ a set of edges, a sequence s_{seq} to align to the graph, its quality values s_{qual} and alignment parameters $match$ (m), $mismatch$ (mm), gap_{open} (q) and gap_{extend} (r)
Output: A POA graph $\mathcal{G}_a = (\mathcal{V}_a, \mathcal{E}_a)$ with incorporated s_{seq} and s_{qual}
Function AlignSequenceToGraphNW($\mathcal{G}, s_{seq}, s_{qual}, m, mm, q, r$) **begin**

1 $b \leftarrow$big negative value
2 $x \leftarrow |\mathcal{V}|$
3 $y \leftarrow |s_{seq}|/8$ ▷ *8 variables per SIMD vector*
4 $\Sigma \leftarrow \{A, C, T, G\}$ ▷ *Alphabet*
5 $QQ \leftarrow [q, q, q, q, q, q, q, q]$
6 $RR \leftarrow [r, r, r, r, r, r, r, r]$
7 $P[|\Sigma|][y] \leftarrow sequenceProfile(s_{seq}, m, mm, \Sigma)$ ▷ *As described in (Rognes 2002)*
8 $H[x][y] \leftarrow$initialize first row vectors for NW alignment, set all other vectors values to b
9 $fcv[y] \leftarrow$initialize first column for NW alignment
10 $F[x][y] \leftarrow$initialize all vectors values to b
11 $M[8] \leftarrow [[b, 0, 0, 0, 0, 0, 0, 0], [b, b, 0, 0, 0, 0, 0, 0], ..., [b, b, b, b, b, b, b, b]]$ ▷ *Masks needed to properly update H*
12 $S \leftarrow [b, b, b, b, b, b, b, b]$ ▷ *Score vector*
13 $max_{score} \leftarrow b$
14 $max_i \leftarrow -1$
15 $topologicalSort(\mathcal{G})$
16 **for** $i = 0$ *to* $(x - 1)$ **do**
17 **for** p *in* $\mathcal{V}[i].predecessors$ **do**
18 $X \leftarrow [fcv[p], 0, 0, 0, 0, 0, 0, 0]$
19 **for** $j = 0$ *to* $(y - 1)$ **do** ▷ *Solve the elements of vertical Gotoh matrix*
20 $F[i][j] \leftarrow$ MAX($F[i][j]$, $H[p][j] + QQ + RR$, $F[p][j] + RR$)
21 $T_1 \leftarrow (H[p][j]$ RSHIFT 7)
22 $T_2 \leftarrow (H[p][j]$ LSHIFT 1) OR X
23 $X \leftarrow T_1$
24 $H[i][j] \leftarrow$ MAX($H[i][j]$, $T_2 + P[\mathcal{V}[i].base][j]$, $F[i][j]$)
25 $E \leftarrow [0, 0, 0, 0, 0, 0, 0, fcv[i]]$
26 **for** $j = 0$ *to* $(y - 1)$ **do** ▷ *Solve the elements of horizontal Gotoh matrix*
27 $E \leftarrow ((H[i][j]$ LSHIFT 1) OR (E RSHIFT 7)) $+ QQ + RR$
28 $T_3 \leftarrow E$
29 **for** $k = 0$ *to* 7 **do** ▷ *Parallelization cannot be used due to dependencies*
30 $T_3 \leftarrow (T_3$ LSHIFT 1) + (RR LSHIFT $(k + 1)$)
31 $E \leftarrow$ MAX(E, $M[k]$ OR T_3)
32 $H[i][j] \leftarrow$ MAX($H[i][j]$, E)
33 $E \leftarrow$ MAX($H[i][j]$, E - QQ)
34 **if** $|\mathcal{V}[i].successors| = 0$ **then**
35 $S \leftarrow$ MAX(S, $H[i][y - 1]$)
36 **if** $max_{score} < maxElement(SCORE)$ **then**
37 $max_{score} \leftarrow maxElement(SCORE)$
38 $max_i \leftarrow i$
39 $\mathcal{A} \leftarrow$ alignment path traced back from the cell with the highest score ($H[max_i][y - 1]$) by checking all possible predecessor cells. As elements from SIMD vectors can not be accessed directly, SIMD vectors are loaded into normal vectors when needed.
40 $AddAlignmentToGraph(\mathcal{G}, \mathcal{A}, s_{seq}, s_{qual})$ ▷ *As described in (Lee et al. 2002)*
41 **return** $\mathcal{G}$

Algorithm 3. Pseudocode for the SPOA algorithm. The displayed function aligns a sequence to a preconstructed POA graph using SIMD intrinsics. Capitalized variables are SIMD vectors. Alignment mode is Needleman-Wunsch.

previous vector for every predecessor (Algorithm 3, lines 21–23), which is needed to compare the upper-left diagonal of the current cell to the cell one row up. Processing the matrix horizontally is not performed using SIMD operations due to data dependencies (each cell depends on the result of the cell to the left of it) and is instead processed linearly (Algorithm 3, lines 25–33). SPOA uses shifting and masking to calculate every particular value of a SIMD vector individually (Algorithm 3, lines 29–31). After the alignment is completed, the traceback is performed (Algorithm 3, line 39) and integrated into the existing POA graph (Algorithm 3, line 40).

SIMD intrinsics decrease the time complexity for alignment from $O((2n_p + 1)n|V|)$ to roughly $O((2n_p/k + 1)n|V|)$, where k is the number of variables that fit in a SIMD vector. SPOA supports

Table 7. Versions of various tools used in this paper

Tool	Commit/version	Source
Racon	2f41352ab4aa on branch *grpaper*	https://github.com/isovic/racon
minimap	1*cd6ae3bc7c7*	https://github.com/lh3/minimap
miniasm	17*d5bd*12290*e*	https://github.com/lh3/miniasm
Canu	*ab*50*ba*3*c*0*cf*0	https://github.com/marbl/canu
FALCON-integrate	8*bb*2737*fd*1*d*7	https://github.com/PacificBiosciences/FALCON-integrate
Sparc	ae89503a1613	https://github.com/yechengxi/Sparc
Nanocorrect	*b*09*e*93772*ab*4	https://github.com/jts/nanocorrect
Nanopolish	47*dcd*7*f*147*c*	https://github.com/jts/nanopolish
MUMmer	3.23	http://mummer.sourceforge.net/

Intel SSE version 4.1 and higher, which embed 128-bit registers. Both short (16 bits) and long (32 bits) integer precisions are supported (therefore k equals eight and four variables, respectively). Eight-bit precision is insufficient for the intended application of SPOA and is therefore not used. Alongside global alignment displayed in Algorithm 3, SPOA supports local and semi-global alignment modes, in which SIMD vectorization is implemented as well.

Implementation and reproducibility

Racon and SPOA are both implemented in C++. All tests were run using Ubuntu-based systems with two six-core Intel Xeon E5645 CPUs at 2.40 GHz with hyperthreading, using 12 threads where possible. The versions of various methods used in the comparisons reported here are shown in Table 7. On PacBio data sets, Canu was run using the "-pacbio-raw" option, FALCON using the config file presented in Supplemental Table S4, and Sparc using "k 1 c 2 g 1 t 0.2". On ONT data sets, Canu was run using the "-nanopore-raw" option, FALCON using the config file presented in Supplemental Table S5, and Sparc using "k 2 g 2 t 0.2". To provide initial alignments for Sparc, BLASR was run using "-bestn 1 -m 5 -minMatch 19". Racon was run with the default parameters on all data sets except *S. cerevisiae* S288c ONT R9, where we increased the base quality value threshold to 20 using the parameter "--bq 20" (the default value is 10). To provide the initial raw assemblies for both Racon and Sparc, minimap (as an overlapper) and miniasm were used. Minimap was run using the "-Sw5 -L100 -m0" options on all data sets (PacBio and ONT); miniasm, with default parameters. For error-correction purposes, Canu was run using "-correct -nanopore-raw", FALCON with an additional config entry "target = preassembly," and Racon with the "--erc" option.

Data sets

Six publicly available PacBio and Oxford Nanopore data sets were used for evaluation. These are listed in Table 8.

Evaluation methods

The quality of called consensus sequences was evaluated primarily using Dnadiff (Delcher et al. 2003). The parameters we took into consideration for comparison include total number of bases in the query, aligned bases on the reference, aligned bases on the query, and average identity. In addition, we measured the time and memory required to perform the entire assembly process by each pipeline.

The quality of error-corrected reads was evaluated by aligning them to the reference genome using GraphMap (Sović et al. 2016b) with the settings "-a anchorgotoh" and counting the match, mismatch, insertion, and deletion operations in the resulting alignments.

Software availability

Racon and SPOA are available open source under the MIT license at https://github.com/isovic/racon and https://github.com/rvaser/spoa. In addition, the version of source codes used in this study can be found in the Supplemental Material.

Table 8. Description of the six data sets used for evaluation

Data set	Description
Lambda phage ONT R7.3	ENA submission ERA476754, with 113× coverage of the NC_001416 reference genome (48,502 bp). Link: ftp://ftp.sra.ebi.ac.uk/vol1/ERA476/ERA476754/oxfordnanopore_native/Lambda_run_d.tar.gz. This data set was subsampled to coverages of 30× and 81× for testing.
E. coli K-12 ONT R7.3	54× pass 2D coverage of the genome (U00096.3, 4.6Mbp). Link: http://lab.loman.net/2015/09/24/first-sqk-map-006-experiment/.
E. coli K-12 PacBio P6C4	160× coverage of the genome (U00096.3). The data set was generated using one SMRT cell of data gathered with a PacBio RS II System and P6-C4 chemistry on a size-selected 20-kbp library of *E. coli* K-12. Link: https://s3.amazonaws.com/files.pacb.com/datasets/secondary-analysis/e-coli-k12-P6C4/p6c4_ecoli_RSII_DDR2_with_15kb_cut_E01_1.tar.gz.
S. cerevisiae S288c ONT R9	59× 2D (pass and fail) coverage of the genome (http://downloads.yeastgenome.org/sequence/S288C_reference/chromosomes/fasta/). Link: http://www.genoscope.cns.fr/externe/Download/Projets/yeast/datasets/raw_data/S288C/ (Istace et al. 2016).
S. cerevisiae PacBio W303 PacBio P4C2	The data set is composed of 11 SMRT cells (https://github.com/PacificBiosciences/DevNet/wiki/Saccharomyces-cerevisiae-W303-Assembly-Contigs), of which one was not used in this study because the containing folder ("0019") was incomplete and the data could not be extracted. The S288C reference (12.1 Mbp) was used for comparison (http://downloads.yeastgenome.org/sequence/S288C_reference/chromosomes/fasta/). Coverage of the data set with respect to the S288C reference is approximately 127×.
C. elegans PacBio P6C4	Bristol mutant strain, 81× coverage of the genome (gi|449020133). The data set was generated using 11 SMRT cells P6-C4 chemistry on a size-selected 20-kbp library. Link: https://github.com/PacificBiosciences/DevNet/wiki/C.-elegans-data-set.

Competing interest statement

M.S. has received a complimentary place at an Oxford Nanopore Technologies organized symposium. N.N. has received travel expenses and accommodation from Oxford Nanopore to speak at two organized symposia.

Acknowledgments

This work has been supported in part by Croatian Science Foundation under the project UIP-11-2013-7353. I.S. is supported in part by the Croatian Academy of Sciences and Arts under the project "Methods for alignment and assembly of DNA sequences using nanopore sequencing data." N.N. is supported by funding from A*STAR, Singapore.

Author contributions: I.S., M.S., and R.V. devised the original concept for Racon. I.S. developed and implemented several prototypes for Racon. R.V. suggested and implemented the SIMD POA version. I.S. implemented a version of Racon that uses SPOA for consensus generation and error correction. I.S. and R.V. tested Racon and SPOA. M.S. and N.N. provided helpful discussions and guidance for the project. M.S. helped define applications and devised tests for the paper; I.S. ran the tests and collected and formatted the results. I.S. and R.V. wrote the paper with input from all authors. M.S. and R.V. designed the figures for the paper. N.N. proof-read and corrected parts of the paper. All authors read the paper and approved the final version. M.S. coordinated the project and provided computational resources.

References

Berlin K, Koren S, Chin C-S, Drake JP, Landolin JM, Phillippy AM. 2015. Assembling large genomes with single-molecule sequencing and locality-sensitive hashing. *Nat Biotechnol* **33:** 623–630.

Chaisson MJ, Tesler G. 2012. Mapping single molecule sequencing reads using basic local alignment with successive refinement (BLASR): application and theory. *BMC Bioinformatics* **13:** 238.

Chin C-S, Alexander DH, Marks P, Klammer AA, Drake J, Heiner C, Clum A, Copeland A, Huddleston J, Eichler EE, et al. 2013. Nonhybrid, finished microbial genome assemblies from long-read SMRT sequencing data. *Nat Methods* **10:** 563–569.

Chin C-S, Peluso P, Sedlazeck FJ, Nattestad M, Concepcion GT, Clum A, Dunn C, O'Malley R, Figueroa-Balderas R, Morales-Cruz A, et al. 2016. Phased diploid genome assembly with single molecule real-time sequencing. *Nat Methods* **13:** 1050–1054.

Delcher AL, Salzberg SL, Phillippy AM. 2003. Using MUMmer to identify similar regions in large sequence sets. *Curr Protoc Bioinformatics* Chapter 10**:** Unit 10.3.

Gotoh O. 1982. An improved algorithm for matching biological sequences. *J Mol Biol* **162:** 705–8.

Istace B, Friedrich A, d'Agata L, Faye S, Payen E, Beluche O, Caradec C, Davidas S, Cruaud C, Liti G, et al. 2016. de novo assembly and population genomic survey of natural yeast isolates with the Oxford Nanopore MinION sequencer. bioRxiv doi: 10.1101/066613.

Koren S, Walenz BP, Berlin K, Miller JR, Phillippy AM. 2017. Canu: scalable and accurate long-read assembly via adaptive *k*-mer weighting and repeat separation. *Genome Res* (this issue). doi: 10.1101/gr.215087.116.

Lee C. 2003. Generating consensus sequences from partial order multiple sequence alignment graphs. *Bioinformatics* **19:** 999–1008.

Lee C, Grasso C, Sharlow MF. 2002. Multiple sequence alignment using partial order graphs. *Bioinformatics* **18:** 452–464.

Li H. 2016. Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences. *Bioinformatics* **32:** 2103–2110.

Loman NJ, Quick J, Simpson JT. 2015. A complete bacterial genome assembled de novo using only nanopore sequencing data. *Nat Methods* **12:** 733–735.

Loose M, Malla S, Stout M. 2016. Real-time selective sequencing using nanopore technology. *Nat Methods* **13:** 751–754.

Myers G. 1999. A fast bit-vector algorithm for approximate string matching based on dynamic programming. *J ACM* **46:** 395–415.

Rognes T, Seeberg E. 2000. Six-fold speed-up of Smith-Waterman sequence database searches using parallel processing on common microprocessors. *Bioinformatics* **16:** 699–706.

Šošić M, Šikić M. 2016. Edlib: A C/C++ library for fast, exact sequence alignment using edit distance. bioRxiv doi: 10.1101/070649.

Sović I, Križanović K, Skala K, Šikić M. 2016a. Evaluation of hybrid and non-hybrid methods for de novo assembly of nanopore reads. *Bioinformatics* **32:** 2582–2589.

Sović I, Šikić M, Wilm A, Fenlon SN, Chen S, Nagarajan N. 2016b. Fast and sensitive mapping of error-prone nanopore sequencing reads with GraphMap. *Nat Commun* **7:** 11307.

Ye C, Ma Z. 2016. Sparc: a sparsity-based consensus algorithm for long erroneous sequencing reads. *PeerJ* **4:** e2016.

Received August 5, 2016; accepted in revised form January 17, 2017.