Technology and Species independent Simulation of Sequencing data and Genomic Variants

Filippo Geraci *
Istituto di Informatica e Telematics
Consiglio Nazionale delle Ricerche
Pisa, Italy
filippo.geraci@iit.cnr.it

Riccardo Massidda
Dipartimento di Informatica
Universitá di Pisa
Pisa, Italy
r.massidda@studenti.unipi.it

Nadia Pisanti
Dipartimento di Informatica
Universitá di Pisa
Pisa, Italy
pisanti@di.unipi.it

***Abstract*—Highly accurate genotyping is essential for genomic projects aimed at understanding the etiology of diseases as well as for routinary screening of patients. For this reason, genotyping software packages are subject to a strict validation process that requires a large amount of sequencing data endowed with accurate genotype information. In-vitro assessment of genotyping is a long, complex and expensive activity that also depends on the specific variation and locus, and thus it cannot really be used for validation of in-silico genotyping algorithms. In this scenario, sequencing simulation has emerged as a practical alternative. Simulators must be able to keep up with the continuous improvement of different sequencing technologies producing datasets as much indistinguishable from real ones as possible. Moreover, they must be able to mimic as many types of genomic variant as possible.**

In this paper we describe *OmniSim*: a simulator whose ultimate goal is that of being suitable in all the possible applicative scenarios. In order to fulfill this goal, *OmniSim* uses an abstract model where variations are read from a .vcf file and mapped into edit operations (insertion, deletion, substitution) on the reference genome. Technological parameters (e.g. error distributions, read length and per-base quality) are learned from real data. As a result of the combination of our abstract model and parameter learning module, *OmniSim* is able to output data in all aspects similar to that produced in a real sequencing experiment.

The source code of *OmniSim* is freely available at the URL: https://gitlab.com/geraci/omnisim

***Index Terms*—NGS sequencing, simulation, genomic variants**

I. Introduction

DNA sequencing is the task of determinating the ordered sequence of nucleotides of a biological sample. In spite of this very simple definition, the last fifty years have witnessed one of the biggest collaborative efforts to improve and refine sequencing technologies [1]. Increasing output quality and throughput has inspired researchers eager for understanding the mechanisms of life, and has fostered the development of astonishing clinical applications. However, to the fast enhancement of technology (in particular the so-called next generation sequencing [2]) did not always follow an equally fast development of algorithms and methods for genomic data analysis. In order to assess variant detection algorithms, designers need large collections of accurate genotype annotations, as well as the actual genomic sequences. Producing such essential supplementary information is still slow and expensive. In this scenario, simulation comes into play, promising a cheap and easy way to carry experiments on a large scale by producing genomic sequences endowed with genotypes. The drawback of using simulators is the need to be confident that results on real data would be comparable with those achieved on simulated ones.

Three generations [3] of continuously evolving sequencing technologies (Sanger, high throughput short reads and long reads), several types of genomic variations (from single nucleotide polymorphisms to structural variants), and different error profiles, actually make the development and validation of accurate simulation pipelines a challenging task.

First generation (Sanger) sequencing produces reads slightly smaller than 1kbp that are merged in larger contigs by means of the shotgun sequencing technique [4]. The low throughput of this technology makes it not suitable for large scale projects but enables very accurate genotypization because of its low error rate.

The market of the second generation of sequencing (sequencing by synthesis) has been dominated by Illumina (with many notable and valuable exceptions). In most applications, reads are produced in pairs, sequencing the two endpoints of long fragments. The read length is much smaller than that of the first generation, and started from about 35bps of the early stage of this technology to one or two hundred bps of its maturity. For this technology, substitutions are the most common type of error and tend to become more frequent in the last reading cycles and the higher error rate is compensated with coverage exploiting the massively higher throughput and (consequent) much lower costs.

The third generation of sequencing technologies has further revolutionized the scenario producing impressively long reads which span a large length range from few kilobases up to more than a hundred of kilobases. Errors are typically more frequent than those of the previous generations, and can more likely include insertions and deletions, making read mapping particularly challenging.

Besides technology-dependent constraints, there are other factors that influence the development of simulators. In particular, handling the largest possible set of variant types would be necessary to make the output more realistic and

* Corresponding author

usable. However, as shown in table 4 of [5], the majority of simulators is limited to few common types or handle only some specific one. There are two alternative approaches for variant generation. The most common method is that of embedding variations at random positions of the reference sequence controlling, feeding the tool with *just* a mutation rate parameter. Often, the spatial distribution across the genome and the type of variant is learned from a real dataset. The other complementary approach generates variants according to a list of types and genomic locations taken as input.

Secondary features of simulators include: the ability to generate datasets for metagenomics, the possibility of introducing PCR-based noise, and the generation of already aligned data.

In this paper we describe *OmniSim*: a novel simulator that - simultaneously - fulfills the promise (i) to be independent from the sequencing technology, and (ii) to handle all the possible types of genomic variant. The technological parameters (e.g. errors and lengths distributions) for our tool are learned by a specialized module that analyzes a sequencing archive in .bam format and extracts information about: read length distribution, error types and probability, quality scores, and PCR noise. Genomic variants are generated according to a user-provided annotation .vcf file. We map each variant into a series of edit operations on the reference genome, so that our simulator can handle every possible type of variant. We also pursue secondary features such as (i) simulating the effect of PCR-based stutter noise in repeated sequences, (ii) enabling variants on different reference genomes to be mixed so as to simulate metagenomics data, and (iii) returning aligned data so as to speed up subsequent experiments.

Experiments, using the paired-end 100bp Illumina platinum genome [6] of the NA12877 human sample and the Oxford Nanopore sequencing [7] of the NA12878 human sample, both from the CEPH pedigree 1463, have shown that *OmniSim* is reasonably fast and completes the entire pipeline (i.e. generation of two human alleles including all the variations reported in dbSNP [8], learning of the error model, and generation of 100 million reads) in about 2/3 hours. Experiments have also shown that the nucleotide composition and the distribution of per-base quality scores of the simulated reads perfectly reflects that of a real sequencing. The source code of *OmniSim* is freely available at the URL: https://gitlab.com/geraci/omnisim

II. Related work

Introducing their work, in [9] the authors complain that simulators are usually designed for specific needs, and are not general enough to be employed in different applicative scenarios. This complaint is somehow confirmed in [5] where the authors provide a decision tree with the purpose of guiding users to an aware choice of the simulator that better fits their needs.

The main aspects on which simulators differ are: the sequencing technologies they can reproduce, the type of variants introduced, the adopted error model, and the richness of the output.

Short reads sequencing is the most represented technology with probably dozens of specialized tools. Among them, we cite GemSIM [10], IntSIM [11], SInC [12], pIRS [13], and ART [14]. Short read simulation tools are usually vendor independent and differ on other features like the error model or the type of variants. In this category, Grinder [15] has the unique feature to be able to accurately simulate shotgun and amplicon sequencing. The tools able to simulate long reads, instead, are usually specialized in either Pacific Bioscience SMRT sequencing or Oxford Nanopore sequencing with the notable exception of SiLiCO [16] that can simulate both of them. Simulators tailored on Pacific Bioscience SMRT sequencing are more common. In particular, the SimLoRD [17] can only simulate *circular consensus sequence reads* (CCS) while PaSS [18] and NPBSS [19] can also produce continuous long reads (CLR). Due to its complexity and rapid evolution, the Oxford Nanopore Sequencing simulation is more challenging and hence less common. In this case, simulators do not produce the sequence of nucleotides but the raw signal emitted by the sequencer. The interesting rationale of this choice is that of opening to the development and testing of base callers. Among the most valuable methods we mention [20] and [21]. It worth citing FASTQSim [22] which is the only attempt to simulate NGS data both for short and long reads (at least PacBio SMRT). Like for our method, this is made possible by learning the parameters from real datasets. However, as shown in [17] figure 1, FASTQSim the error rate does not agree well with the raw data.

Handling metagenomics has also emerged as a useful feature of simulators. This task is complicated by the fact that reads come from several organisms, and hence, the generation needs to be able to deal with large data. Specific tools have been implemented for this task: MetaSim [23] is probably the first attempt in this sense; it can simulate either Sanger sequencing or sequencing by synthesis. The tool BEAR [9] focuses on simulating a realistic abundance of samples without the need of learning it from real data; GemSIM [10] implements an accurate statistical model to handle sequencing errors but it only simulates single nucleotide polymorphisms (SNPs) variants. In our case, as we will explain in detail in section III, metagenomics simulation is achieved allowing alleles generated from different organisms to be mixed before reads simulation.

Mostly, simulators produce their output in fasta or fastq format, thus losing the alignment information. This has the disadvantage of leaving to the user the burden of alignment. An interesting exception is [24] that optionally returns also the aligned reads using the standard .bam format. Moreover, producing fastq does not necessarily mean producing accurate per-base qualities. Grinder [15], for example, produce qualities with a fixed value. *OmniSim* has an option to keep track of the genomic position of the generated reads so as to produce the alignment in .bam format. Moreover, the per-base quality is first learnt from real data and then produced in accordance with the simulated sequencing errors.

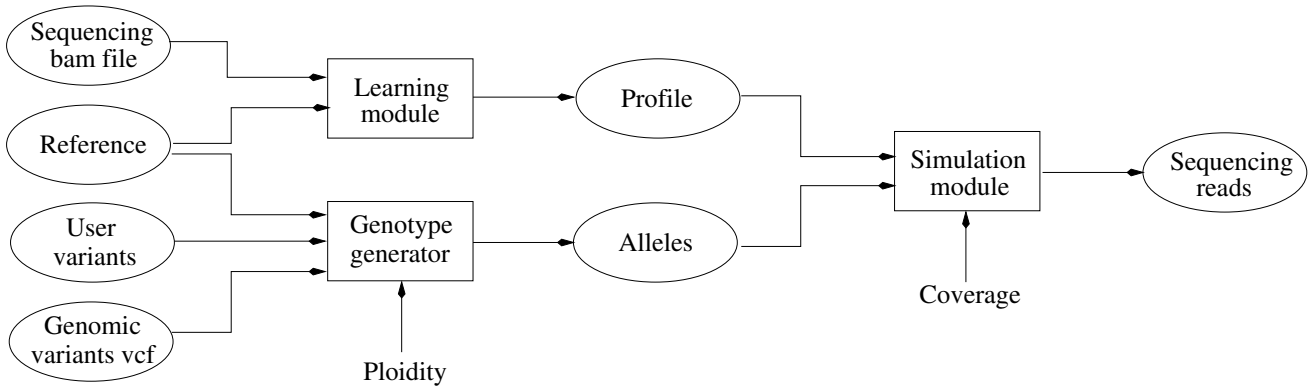


Fig. 1. Structure of the simulation pipeline: the three modules of our tool are in rectangular boxes

III. Method

Our simulator consists of the three modules depicted in Figure 1:

1) A *Genotype generator* that takes in input a reference genome and a list of variants, and produces a customizable number of alleles;
2) A *Learning module* that extracts from a .bam file the technological parameters (e.g. distributions of lengths, errors, etc.), and
3) The *Simulation module* that actually generates the sequences from a set of alleles (e.g. coming from the first module) according to a given profile (which can be, for example, the outcome of the learning module).

The three modules are designed to be as independent from each other as possible, in order to be possibly employed for diverse usage scenarios. For example, developing de-novo genome assemblers, it is possible to simulate the sequencing of the same individual with different technologies by generating the reads from the same set of alleles but changing the technological parameters.

A further possible advantage of having the reads production independent from the genotype generator, is that the genotypes of different species can be mixed for metagenomics simulations purposes.

A. Genotype generation

Current simulators have two main drawbacks: they do not support all the possible types of genomic mutation at once (see table 4 in [5]), and variations are spread across the genome according to predetermined statistical distributions that not necessarily reflect the real distribution for the species of interest. The first issue causes both the generation of unrealistic reads (where only a subset of variations is present) and the need to perform separate validations using a suitable simulator per each type of variation the user is interested in.

The second issue can become relevant when testing algorithms for which the distribution of the variations can affect the output. For example, as explained in [25], in the single individual haplotyping problem (SIH) the distance between consecutive SNPs can let some algorithms fail to reconstruct the correct haplotype.

The goal of the genotype generation module is that of overcoming both limitations. We started from the practical evidence that, for the large majority of the most commonly studied organisms, a reasonably complete map of the genomic variations with their type, estimated frequency, and exact position, is available in public databases like dbSNP [8]. For these organisms, maps can be used, rather than leveraging on a statistical model to decide the position, type and content of variations.

We further observed that all genomic variants are alterations of the DNA sequence that can be modeled as a series of unary edit operations (insertion, deletion, replacement). Therefore, variants types can be mapped into a suitable sequence of edit operations whose final outcome corresponds to them, and hence their application can be simulated by applying those.

The application of a specific variant on an allele is not always guaranteed because of uncontrollable aspects such as the allelic frequency and the previous application of another conflicting variant. Conflicts arise when two variants insist on the same locus. A typical example of this situation is that of a SNP taking place in the same DNA portion of - say - a deletion: the latter let the nucleotide involved in the SNP disappear. Another case is when the same locus undergoes both an insertion and a deletion: although these variations could theoretically co-exist, their concurrent application would probably not make sense, and thus only one variation should be considered. According to different applicative scenarios, users may - or may not - want to leave to the faith the decision of which of the alternative variations to apply (either forcing a specific one to be selected regardless of the others or making a random choice). Forcing a specific variant is the case, for example, of [26] where the same tandem repeat is increasingly extended in order to evaluate the longest genotypable region.

Our genotype generation module takes as input a reference genome, a map (as a .vcf file) of all the annotated variants, and a list of user-defined variants, and returns a (customizable) number of allele sequences with the corresponding alignment to the reference genome. The allele' generation procedure

iterates over the chromosomes copying the reference sequence until the position of a genomic variant is reached. The variation is applied to a randomly selected allele according to its observed frequency. Possible subsequent mutations on the same locus are applied to other alleles (again, according to their observed frequencies) until when all the alleles have been generated. If no variants are selected for an allele, then the reference sequence is just copied. Before applying a mutation to an allele, the procedure checks whether there is a conflict with previously introduced variants (this check is done by verifying that the allele's sequence matches the reference in the positions involved in the new mutation). In order to force user-defined variants, these mutations are set to omnipresent in the population (namely, observed with probability 1), and are evaluated before the others (so that they cannot be withdrawn because of conflicts). Once a mutation is introduced into an allele, the corresponding alignment sequence is updated.

Although fast, simulating a population with different genotypes, the above procedure could need to be run a large number of times. In this case, exploiting the CPU parallelism to speed up the alleles' generation would be useful. We achieve this goal by allowing the parallel generation of distinct chromosomes.

B. Parameter learning

Sequencing platforms produce reads with very different characteristics in terms of read length distribution, error types, and coverage. All these parameters need to be learned in order to accurately simulate the desired sequencing procedure. Interestingly, none of them depends on the organism under study, and thus it is possible to simulate sequencing of any sample regardless of the organism used to learn parameters.

In order to collect statistics useful to mimic the behavior of a given technology, the learning module leverages on a reference genome and a set of aligned reads in .bam format. Both single-end and paired-end are supported. In the latter case, the error distribution and types are learned separately from the two mate reads, since it is a common experience that the two quality profiles can differ.

Whole genome mapping software packages such as BWA [27] are designed to provide suboptimal alignments sacrifying accuracy for speed. For applications where the optimal alignment is required, specialized tools like the Genome Analysis Toolkit (GATK) [28] can be used to post-process .bam files and refine the reads alignment. Following a similar philosophy, our parameter learning module re-aligns reads onto the reference by means of the ultrafast bit-vector-based semi-global alignment algorithm described in [29] and implemented in [30]. We call non-matching characters in this alignment as sequencing errors. At first sight, this could be considered as a rough simplification since insertions, deletions and mismatches could be due to genomic variations rather than to errors. In practice, we observed that the higher probability of errors (compared to that of variations) makes this approximation acceptable. However, in order to reduce the bias due to misinterpretation of variants, in the genotype generation module we implemented a function to return a (multiploid) reference with the most probable alleles. This reference can be used as input for the parameter learning step, in order to reduce the possible wrong contribution of variants. When a read is processed, the module aligns it with all the alleles of the reference, and then decides for the one that maximizes the alignment score.

Due to the technology-specific distribution and type of errors, the learning module collects separate statistics for each position of the alignment string, as well as for each type of error. This approach allows to obtain a more accurate and realistic simulation. For example, in short-reads sequencing by synthesis, the probability of erroneously calling T and C as G increases as the chemistry degrades with the succession of cycles. In particular, we keep and update a *transition table* that stores the conditional probability of encountering a certain alignment symbol (match, mismatch, insert, delete, end of sequence), given the preceding position symbol. Moreover, we maintain a histogram for each alignment symbol/position with the distribution of the per-base quality score.

Mismatches require to decide which character to use for replacement. As mentioned above, substitutions are not equiprobable, and depend on several technological factors like the sequencing machinery and chemistry. Although a random choice of the replacement character would still be appropriate for most applications, it might not produce realistic results deceiving software for systematic bias removal (see [31] for an in-depth discussion on these tools). In order to drive the correct replacement, we maintain a (positional) *substitution table* with the probability of each base to be replaced with another.

Unlike deletions that do not require any additional information, insertions need a mechanism to produce the sequence to embed in the read at the simulation stage. Keeping track of sequences involved in insertion events would not be enough for such a mechanism since they depend on several contextual variables including the neighborhood. In this case, however, almost all the applications just need that the embedded string causes the introduction of a gap in the alignment of the reads. This can easily be achieved ensuring to insert immediately after the insertion point a string that completely mismatches the reference.

Although the above statistics would be enough to generate technology-specific reads, they do not capture parameters that are dependent on the specific biases introduced with library preparation. The two most important of these parameters are: stutter noise (introduced with PCR amplification), and fragment size distribution for paired-end sequencing.

Stutter noise introduces/removes spurious units on microsatellites (aka short tandem repeats). The entity of this phenomenon is more dependent on the unit length than on its content; thus, in order to quantify it, for each possible unit length (microsatellites are defined in the range from 1bp to 6bp), we keep track of the discrepancy in the number of units between pure tandem repeats located in the reads and the corresponding sequence in the reference.

When dealing with paired-end sequencing, the mate reads

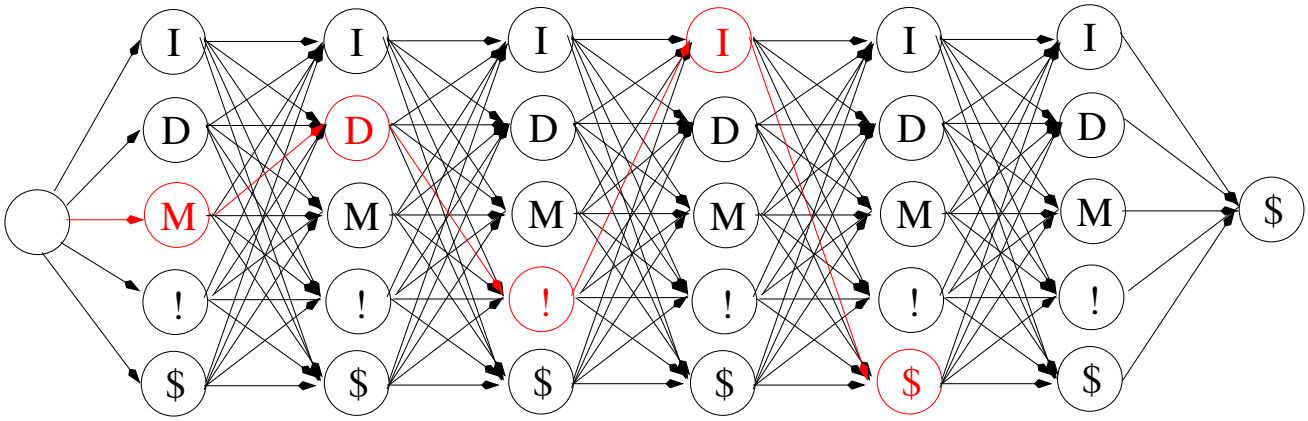


Fig. 2. Example of multi-partite graph able to generate an alignment string up to 6 characters. M is for matches, ! for mismatches, I for insertions, D for deletions and $ for the end of sequence. In red the example alignment that generate the sequence MD!I.

are derived from the two endpoints of a (possibly longer) DNA fragment. According to the experimental setup, some part of the sequence might either be read twice or not be sequenced. For a more accurate mimic of the DNA fragmentation, we collect statistics on the distribution of the insert sizes, as well as we store the information about the orientation the paired reads.

C. Reads generation

The read generation module takes as input the alleles (produced by the genotype generator) and a profile of sequencer, and returns a set of reads either in fastq or bam format. Alleles derived from different organisms and different executions of the genotype generator can be mixed in order to simulate metagenomics.

Conceptually, the reads simulation mimics the process of library preparation and sequencing. Given a chromosome, the module performs four steps

1) It selects one allele at random,
2) It partitions it into fragments,
3) It introduces PCR artifacts,
4) It extracts from each fragment the corresponding read(s).

Per each chromosome, these steps are repeated until the desired user-defind average coverage depth is reached.

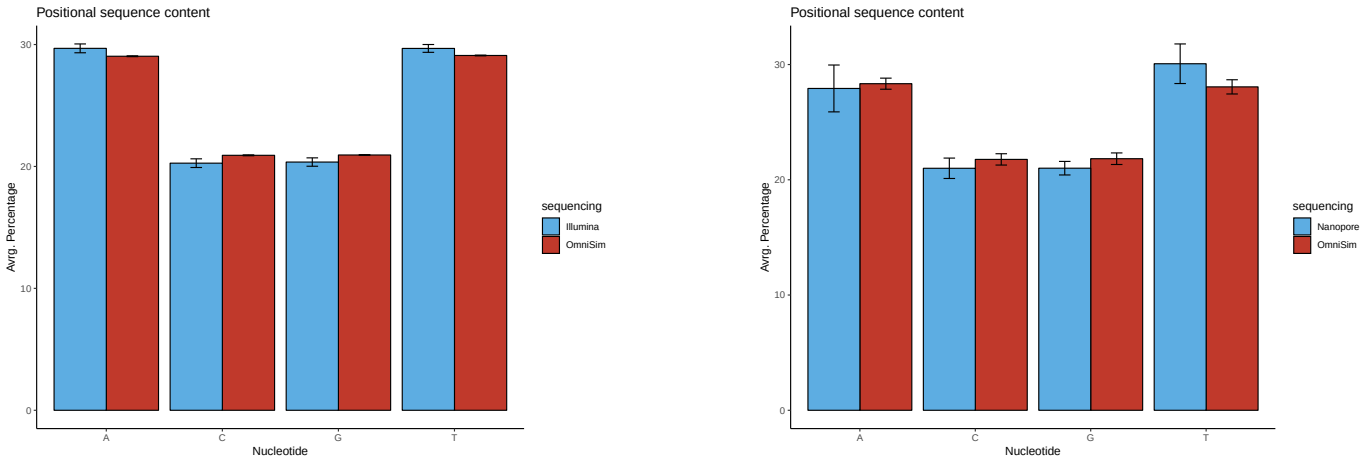
Unfortunately, in real sequencing experiments, allelic frequencies of variations are neither equiprobable nor constant. This is due to two factors: first, the amount of DNA extracted for each allele cannot be tightly controlled and, secondly, fragments size selection causes coverage fluctuations. We simulate the differences of DNA amount modeling the selection of the alleles as a Bernoulli process of probability $1/p$ where p is the number of alleles. Coverage fluctuations are then obtained simulating fragmentation.

Fragmentation depends on the type of sequencing. For single-end sequencing, only the size of the sequenced portion can be known, while it is impossible to determine the real length of segments, and therefore, in this case we do not actually perform the fragmentation, but we rather constrain fragments to be as long as the corresponding read. On the other hand, for paired-end sequencing, reads are extracted from the endpoints of a long fragment whose length can be measured after the alignment with the reference. Hence, we can learn the distribution of fragment lengths in the learning phase, and then perform the fragmentation accordingly.

Before producing the read(s), fragments are preprocessed to introduce a bias similar to that of stutter noise due to PCR amplification [32]. This technique, used during library preparation, tends to add or delete copies of the repeated unit of short tandem repeats deceiving tandem repeat genotyping algorithms. Pure tandem repeats are pinpointed through a simple regular-expression-based procedure.

Reads generation has the twofold goal of extracting the sequence of the nucleotides of the read and injecting sequencing errors. As mentioned in section III-B, these errors are modeled as non-matching characters in the alignment of the read with the corresponding allele. We thus generate a plausible alignment to drive the extraction of correct nucleotides from the reference, and errors from the parameters learned in the previous phase. The alignment symbols are generated as the sequence of nodes belonging to a random walk on a multi-partite graph, where the probability of passing from a node of the k-th independent set to a node of the $k+1$-th independent set is extracted from the k-th *transition table* learned in the previous step. The process stops when an *end of sequence* symbol is generated. Figure 2 shows a small multi-partite graph where an example of random walk and the corresponding alignment are highlighted.

The process of read creation consists of looping over the symbols of an allignment, copying the sequence of nucleotides from the endpoint of a fragment, or inserting an error. There are three possible types of errors: deletions, mismatches, and insertions. Deletions are the easiest type of mutation to cope with, since they do not require any supplementary information: when a deletion symbol is found, the corresponding character in the reference is just skipped. Mismatches require to select a replacement character: this selection is



(a) Illumina

(b) Oxford Nanopore

Fig. 3. Composition of reads

done by means of the learned *substitution table*. In order to model specific phenomena (e.g. polyG tails), we maintain a different substitution table for each possible position in the read. Insertions are complicated by the difficulty of generating the "most appropriate" sequence to be inserted in the read. An approach similar to that we used for mismatches would not work properly, as the inserted sequence depends on too many parameters (i.e. the length of the insertion, its position on the read, the neighborhood, etc.), and keeping track of all of them would require a huge training set. We observed that a "good" insertion sequence should force an alignment software to open a gap. This can easily be achieved by inserting a random sequence provided this does not match any neighborhood of the insertion in the reference.

The read generation process is iterated over all the chromosomes until the global average coverage is reached. As it is the case for real sequencing, the random length of the fragments/reads causes fluctuations of the local coverage, and it leaves open the possibility for certain loci to be under/over represented.

IV. Results

In this section, we describe our experiments aimed at showing that *OmniSim* is able to produce realistic reads for different sequencing technologies while being fast enough to be used in practice. We used a workstation equipped with an Intel(R) Core(TM) i5-3570 CPU @ 3.40GHz and 8Gb of RAM, running Linux Fedora 29 (Workstation Edition). In order to provide an accurate estimation of the overall running time, all the tests have been run using only a single thread.

Our tests consisted of generating two realistic alleles for a human sample, learning a model for an Illumina paired-end 100bp reads sequencing and a model for an Oxford Nanopore sequencing, and finally producing 100 million of synthetic reads.

We Downloaded the latest snapshot of dbSNP [8] (build 152) consisting in a 14Gb gzip compressed .vcf file annotating 690M variants (including alternative variants of the same locus) and used it to generate two alleles from the *hg38.p12* reference (accession GCF_000001405.38).

As for the model we used:

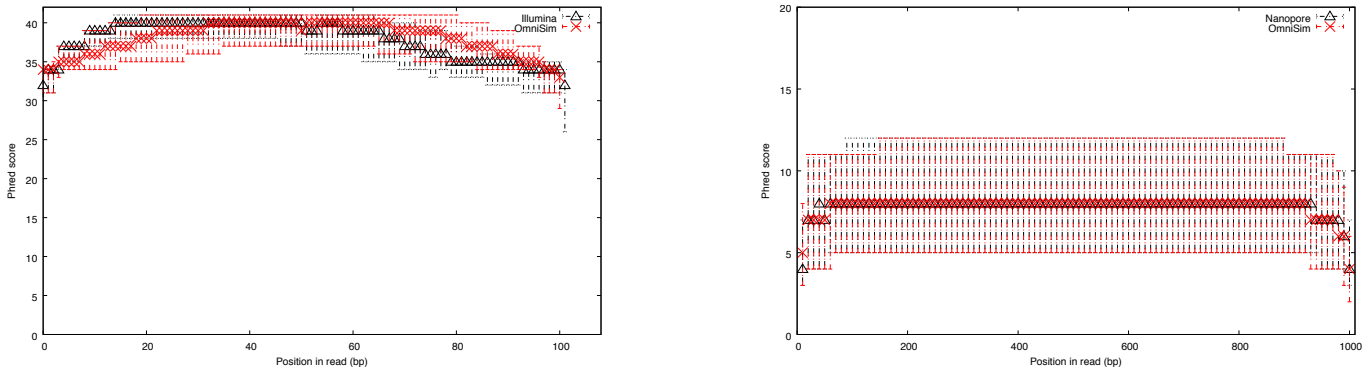
- a uniform subsample of 164.09 million of paired-end reads of length 100bp (14Gb) from the Illumina platinum genomes [6] of the CEPH1463 NA12877 individual (sample accession SAMEA1573614)
- a uniform subsample of 7.95 million of single-end reads (17Gb) of variable length (up to 1kbp) from an Oxford Nanopore sequencing [7] of the CEPH1463 NA12878/GM12878 individual.

Both the NA12877 and NA12878 samples have been aligned to the *hg38* (GCA_000001405.15) primary build with BWA-MEM.

TABLE I
Running time of the three modules

Module	***Illumina pe***	***Nanopore se***
Alleles generation	40m.50s	
Error model learning	1h:07m.05s	8m.31s
Production of 100k reads	16m.16s	1h:25m.05s

In Table I we report the running time of the three modules depicted in Figure 1. Alleles generation is independent of the simulated sequencing technology, but may need to be repeated several times when simulating a population. For Illumina reads, this could cause the first step of the pipeline to take a non-negligible fraction of the overall running time. However, it worth noting that the generation of the alleles has only short-range dependencies and, thus, this module can be easily accelerated by running parallel instances on different chromosomes. The running time of the learning module does not depend on the size (in bp) of the sequences, but rather on their number. In fact, the two samples we used for learning have roughly the same size (although a very different number



(a) Illumina

(b) Oxford Nanopore

Fig. 4. Comparison of the error profile of a real sequencing and a simulated one

of sequences) and the NA12878 has longer reads but smaller running time. This result is not surprising, given the linear time cost of the bit-vector-based semi-global alignment algorithm described in [29]. When generating the Illumina profile, most of the time is spent by edlib [30] for the initialization of the data structure, and this cost is not amortized because of the little length of the reads. Reads generation running time is deeply affected by the length of the reads. In fact, most of the time is spent in the generation of the alignment sequence shown in Figure 2. However, even in this case the module can easily be accelerated by a parallel computation of multiple alignments. Summing up, as shown in table I, the entire pipeline of *OmniSim* can be run in a reasonable time independently from the simulated sequencing technology.

We have investigated the composition of the generated reads and compared it with that of a real sequencing. To this end, we computed the positional distribution of nucleotides and reported the average and standard deviation. As shown in Figure 3, *OmniSim* has the same average distribution of the corresponding sequencing but with a slightly lower standard deviation (notice that the lower standard deviation is due to the smaller number of generated reads compared to the real ones). Considering the uneven distribution of nucleotides on the human reference and the biases of sequencing technologies that tend to under-represent CG-rich regions, this result suggests that *OmniSim* was able to learn these biases and generate reads whose coverage distribution reflects that of a real sequencing. Comparing Illumina sequencing and Oxford Nanopore, we observed that the natural higher variability of the latter is preserved by *OmniSim*. In fact, the standard deviation of the simulated Nanopore sequencing is higher than that of Illumina with the same proportions of the real ones.

Finally, Figure 4 reports the positional average value of the quality scores (and the first and third quartile on the error bar) for both the real and the simulated reads. As for Figure 3 (b), in order to enhance visualization, we grouped the measures in buckets of 10bps. Figure 4 (a) shows that *OmniSim* was able to learn the typical error distribution of Illumina reads: few initial bases with increasing quality, a long plateau of high quality, and a 20bp-long slope with decreasing quality values. Figure 4 (b), instead, shows a comparison of a real Oxford Nanopore sequencing and *OmniSim*. Interestingly, in this case our simulator was not only able to learn the distribution of the average quality values, but also to mimic the variability generating exactly the same standard deviation.

V. Conclusions

In-silico simulation of sequencing has emerged as a promising practical tool to produce genomic data provided with accurate genotype annotations, in order to be used as a validation benchmark. This data is thus essential to drive bioinformaticians during the development and testing of new analysis methods. In the daily practice, however, developers have to face the fact that simulators are usually the result of specific needs, and they are hence far from general. Moreover, it is not uncommon for older simulators not to be maintained or updated, with the practical result of producing reads that no longer reflect the current technology. Selecting the simulator that better fits the user needs, has become so complex that the authors of the survey in [5] ended up providing a decision tree to guide developers.

In this paper we presented *OmniSim*: a new simulation tool that is specifically designed to be independent on the sequencing technology and type of genomic variant. *OmniSim* uses a dedicated module to learn the technological parameters of the sequencing machinery. This allows to keep our simulator up-to-date by simply re-running the learning module on the sequencing data of the latest technology. Genomic modifications are mapped into sequences of edit operations on a reference, enabling *OmniSim* to handle every possible type of structural variant. Experiments on two real datasets (one paired-end 100bp Illumina sequencing, and one Oxford Nanopore sequencing) have demonstrated that *OmniSim* is able to accurately simulate very different sequencing technologies producing reads with the same nucleotide composition and error profile of the real ones. *OmniSim* has not demanding hardware requirements and can run on a relatively small work-

station accomplishing the entire pipeline (alleles generation, error profile learing, reads generation) in a few hours.

REFERENCES

[1] J. M. Heather and B. Chain, "The sequence of sequencers: The history of sequencing dna," *Genomics*, vol. 107, no. 1, pp. 1–8, 2016.

[2] S. Goodwin, J. D. McPherson, and W. R. McCombie, "Coming of age: ten years of next-generation sequencing technologies," *Nature Reviews Genetics*, vol. 17, no. 6, p. 333, 2016.

[3] A. Magi, N. Pisanti, and L. Tattini, "The source of the data flood: Sequencing technologies," *ERCIM News*, vol. 2016, no. 104, 2016.

[4] S. Anderson, "Shotgun dna sequencing using cloned dnase i-generated fragments," *Nucleic acids research*, vol. 9, no. 13, pp. 3015–3027, 1981.

[5] M. Escalona, S. Rocha, and D. Posada, "A comparison of tools for the simulation of genomic next-generation sequencing data," *Nature Reviews Genetics*, vol. 17, no. 8, p. 459, 2016.

[6] M. A. Eberle, E. Fritzilas, P. Krusche, M. Källberg, B. L. Moore, M. A. Bekritsky, Z. Iqbal, H.-Y. Chuang, S. J. Humphray, A. L. Halpern *et al.*, "A reference data set of 5.4 million phased human variants validated by genetic inheritance from sequencing a three-generation 17-member pedigree," *Genome research*, vol. 27, no. 1, pp. 157–164, 2017.

[7] 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.

[8] S. T. Sherry, M.-H. Ward, M. Kholodov, J. Baker, L. Phan, E. M. Smigielski, and K. Sirotkin, "dbsnp: the ncbi database of genetic variation," *Nucleic acids research*, vol. 29, no. 1, pp. 308–311, 2001.

[9] S. Johnson, B. Trost, J. R. Long, V. Pittet, and A. Kusalik, "A better sequence-read simulator program for metagenomics," in *BMC bioinformatics*, vol. 15, no. 9. BioMed Central, 2014, p. S14.

[10] K. E. McElroy, F. Luciani, and T. Thomas, "Gemsim: general, error-model based simulator of next-generation sequencing data," *BMC genomics*, vol. 13, no. 1, p. 74, 2012.

[11] X. Yuan, J. Zhang, and L. Yang, "Intsim: An integrated simulator of next-generation sequencing data," *IEEE Transactions on Biomedical Engineering*, vol. 64, no. 2, pp. 441–451, 2016.

[12] S. Pattnaik, S. Gupta, A. A. Rao, and B. Panda, "Sinc: an accurate and fast error-model based simulator for snps, indels and cnvs coupled with a read generator for short-read sequence data," *BMC bioinformatics*, vol. 15, no. 1, p. 40, 2014.

[13] X. Hu, J. Yuan, Y. Shi, J. Lu, B. Liu, Z. Li, Y. Chen, D. Mu, H. Zhang, N. Li *et al.*, "pirs: Profile-based illumina pair-end reads simulator," *Bioinformatics*, vol. 28, no. 11, pp. 1533–1535, 2012.

[14] W. Huang, L. Li, J. R. Myers, and G. T. Marth, "Art: a next-generation sequencing read simulator," *Bioinformatics*, vol. 28, no. 4, pp. 593–594, 2011.

[15] F. E. Angly, D. Willner, F. Rohwer, P. Hugenholtz, and G. W. Tyson, "Grinder: a versatile amplicon and shotgun sequence simulator," *Nucleic acids research*, vol. 40, no. 12, pp. e94–e94, 2012.

[23] D. C. Richter, F. Ott, A. F. Auch, R. Schmid, and D. H. Huson, "Metasima sequencing simulator for genomics and metagenomics," *PloS one*, vol. 3, no. 10, p. e3373, 2008.

[16] E. A. G. Baker, S. Goodwin, W. R. McCombie, and O. M. Ramos, "Silico: a simulator of long read sequencing in pacbio and oxford nanopore," *BioRxiv*, p. 076901, 2016.

[17] B. K. Stöcker, J. Köster, and S. Rahmann, "Simlord: simulation of long read data," *Bioinformatics*, vol. 32, no. 17, pp. 2704–2706, 2016.

[18] W. Zhang, B. Jia, and C. Wei, "Pass: a sequencing simulator for pacbio sequencing," *BMC Bioinformatics*, vol. 20, no. 1, p. 352, 2019.

[19] Z.-G. Wei and S.-W. Zhang, "Npbss: a new pacbio sequencing simulator for generating the continuous long reads with an empirical model," *BMC bioinformatics*, vol. 19, no. 1, p. 177, 2018.

[20] Y. Li, R. Han, C. Bi, M. Li, S. Wang, and X. Gao, "Deepsimulator: a deep simulator for nanopore sequencing," *Bioinformatics*, vol. 34, no. 17, pp. 2899–2908, 2018.

[21] C. Rohrandt, N. Kraft, P. Gießelmann, B. Brändl, B. M. Schuldt, U. Jetzek, and F.-J. Müller, "Nanopore simulation–a raw data simulator for nanopore sequencing," in *2018 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. IEEE, 2018, pp. 1–8.

[22] A. Shcherbina, "Fastqsim: platform-independent data characterization and in silico read generation for ngs datasets," *BMC research notes*, vol. 7, no. 1, p. 533, 2014.

[24] D. Earl, K. Bradnam, J. S. John, A. Darling, D. Lin, J. Fass, H. O. K. Yu, V. Buffalo, D. R. Zerbino, M. Diekhans *et al.*, "Assemblathon 1: a competitive assessment of de novo short read assembly methods," *Genome research*, vol. 21, no. 12, pp. 2224–2241, 2011.

[25] P. Edge, V. Bafna, and V. Bansal, "Hapcut2: robust and accurate haplotype assembly for diverse sequencing technologies," *Genome research*, vol. 27, no. 5, pp. 801–812, 2017.

[26] H. Tang, E. F. Kirkness, C. Lippert, W. H. Biggs, M. Fabani, E. Guzman, S. Ramakrishnan, V. Lavrenko, B. Kakaradov, C. Hou *et al.*, "Profiling of short-tandem-repeat disease alleles in 12,632 human whole genomes," *The American Journal of Human Genetics*, vol. 101, no. 5, pp. 700–715, 2017.

[27] H. Li and R. Durbin, "Fast and accurate short read alignment with burrows–wheeler transform," *bioinformatics*, vol. 25, no. 14, pp. 1754–1760, 2009.

[28] A. McKenna, M. Hanna, E. Banks, A. Sivachenko, K. Cibulskis, A. Kernytsky, K. Garimella, D. Altshuler, S. Gabriel, M. Daly *et al.*, "The genome analysis toolkit: a mapreduce framework for analyzing next-generation dna sequencing data," *Genome research*, vol. 20, no. 9, pp. 1297–1303, 2010.

[29] G. Myers, "A fast bit-vector algorithm for approximate string matching based on dynamic programming," *Journal of the ACM (JACM)*, vol. 46, no. 3, pp. 395–415, 1999.

[30] M. Šošić and M. Šikić, "Edlib: a c/c++ library for fast, exact sequence alignment using edit distance," *Bioinformatics*, vol. 33, no. 9, pp. 1394–1395, 2017.

[31] D. Laehnemann, A. Borkhardt, and A. C. McHardy, "Denoising dna deep sequencing datahigh-throughput sequencing errors and their correction," *Briefings in bioinformatics*, vol. 17, no. 1, pp. 154–179, 2015.

[32] R. A. Aponte, K. B. Gettings, D. L. Duewer, M. D. Coble, and P. M. Vallone, "Sequence-based analysis of stutter at str loci: Characterization and utility," *Forensic Science International: Genetics Supplement Series*, vol. 5, pp. e456–e458, 2015.