Proceedings of the 2017 Winter Simulation Conference
W. K. V. Chan, A. D'Ambrogio, G. Zacharewicz, N. Mustafee, G. Wainer, and E. Page, eds.

TOWARDS RAPID POPULATION GENETICS FORWARD-IN-TIME SIMULATIONS

Victor Sepulveda

CeBiB, Centre for Biotechnology and Bioengineering
Santiago,8330880, CHILE

Roberto Solar
Alonso Inostrosa-Psijas

CITIAPS, Universidad de Santiago de Chile
Santiago, 8330880, CHILE

Veronica Gil-Costa

CeBiB, CONICET, UNSL
San Luis, 5700, ARGENTINA

Mauricio Marin

CeBiB, Centre for Biotechnology and Bioengineering
DIINF, Universidad de Santiago de Chile
Santiago, 8330880, CHILE

ABSTRACT

Computer simulations are an important tool for the current research in population and evolutionary genetics. They help to understand the genetic evolution of complex processes dynamics that cannot be analytically predicted. The basic idea is to generate synthetic data sets of genetic polymorphisms under user-specified scenarios describing the evolutionary history and genetic architecture of a species. In this work, we focus on forward-in-time simulations which represent the most powerful, but, at the same time, most compute-intensive approach for simulating the genetic material of a population. We present a highly-optimized forward-in-time simulation library called `Libgdrift`, specially designed to create large sets of replicated simulations. Our simulation library uses code optimizations such as spatial locality and a two-phase data compression approach which allow fast simulation executions, while reducing memory storage. Results show that our proposal can improve the performance reported by well-known simulation software.

1 INTRODUCTION

Computer simulations allow us to understand the elements and patterns that can alter a system and can be used to study complex processes, including those that are analytically intractable. Furthermore, the simulation of multiple stochastic replicates may provide the variability required to study different processes, such as the genetic variation in natural populations. In this context, simulation is a fundamental tool for analyzing how basic evolutionary forces such as natural selection, recombination, and mutation can shape the genetic landscape of a population.

There are three basic simulation approaches commonly used for simulating evolutionary population and evolutionary genetics: (1) *backward-in-time* also known as coalescent theory, (2) *forward-in-time* also known as individual-based simulations (usually driven by genetic drift algorithms), and (3) *resampling* (Yuan et al. 2012). The coalescent theory (Kingman 1982) propose that the genetic sequences of a given population with some characteristics like genetic variability and neutral genes, should have had only one common ancestor. Thus, this approach starts from the observed sample of the population in the present and works backwards to infer the genetic history of the population. On the other hand, the forward-in-time simulations focuses on individuals. Everyone in the simulated population undergoes a life cycle: birth, selection, mating, reproduction, mutation, migration and death. The simulation starts from an initial ancestral population and then tracks the evolution of the individuals of the population generation by generation under various demographic and genetic forces, such as: mutation, selection, recombination,

978-1-5386-3428-8/17/$31.00 ©2017 IEEE

fluctuations in the population size and migration. Finally, the resampling approach usually does not require to manage evolution and demography, it randomly generates samples from existing data sets.

In this work, we focus on forward-in-time simulations driven by genetic drift algorithms, because it allows to simulate genetic samples under complex realistic demographic scenarios. In other words, forward-in-time simulations are not restricted by any assumption. Genetic drift is one of the basic mechanisms of evolution. It describes random fluctuations in the numbers of gene variants in a population. It is used to represent the situation when some individuals may - by chance - leave behind a few more descendants than other individuals. Then it can provoke large changes in populations over a short period of time.

We present the `Libgdrift` library which aims to reduce the running time and the amount of memory required to run a forward-in-time simulation. Our optimized library is based in a two-phase compression approach. Compressing consists of encoding a sequence as a concatenation of sub-sequences from a given reference sequence. Then, in the first phase we apply a quaternary base conversion to allocate four DNA nucleotides per byte. In the second phase, given a reference sequence, instead of copying the reference sequence to each individual of the population, we accumulate its mutations. Moreover, our proposed library is optimized to take advantage of data locality. To evaluate the `Libgdrift` library we simulate Wright-Fisher model where each individual has one gene, and at every generation the population dies but another one is born at the beginning of the next generation, so the population size remains stable. We compare the performance achieved by our proposal to other well-known simulation software. Results show that the `Libgdrift` library can reduce the simulations running times by 20% in the worst case (when the mutation rate is too low, about 1e-9).

The remaining of the paper is structured as follows. Section 2 provides related works. In Section 3, we present the `Libgdrift` simulation library. Section 4 describes the two-phase compression approach used in the library. Section 5 presents the experimental results and Section 6 presents concluding remarks.

2 RELATED WORKS

2.1 Forward-in-time Population Genetics Simulations

There are various forward-in-time population genetics simulators presented in the technical literature like the `Fwdpp`, `AnA-FiTS`, `Forqs`, `SLiM` and `SLiM 2`, `AdmixSim` and `XSim` among others. `AnA-FiTS` (Aberer and Stamatakis 2013) is a highly-optimized forward-in-time simulator software of population genetic datasets. With `AnA-FiTS`, neutral mutations are produced at the end of the forward-in-time simulation by keeping a track of the entire ancestry of all surviving individuals on a graph structure.

`Forqs` (Kessner and Novembre 2014) is a forward-in-time simulation of recombination, quantitative traits and selection. `Forqs` uses a haplotype-based approach, i.e. instead of using a mutation-centric approach (keeping track of single-site variants), keeps tracks of individual haplotype chunks as they recombine over multiple generations.

`GeneEvolve` (Tahmasbi and Keller 2017) is a user-friendly and efficient population genetics simulator that handles complex evolutionary and life history scenarios. It generates individual-level phenotypes and realistic whole genome sequence or SNP data. `GeneEvolve` is a forward-in-time simulator which provides a wide range of scenarios for mating systems, selection, population size and structure, migration, recombination, and environmental effects.

`AdmixSim` (Yang et al. 2016) is a forward-in-time simulator based on the Wright-Fisher model, capable of simulating admixed populations with: 1) multiple ancestral populations; 2) multiple waves of admixture events; 3) fluctuating population size; and 4) fluctuating admixture proportions. `XSim` (Cheng, Garrick, and Fernando 2015) is a software developed to efficiently simulate sequence data in descendants of arbitrary pedigrees, and `XSim` implements a strategy to drop-down origins and positions of chromosomal segments.

Fwdpp (Thornton 2014) is a C++ library intended to facilitate the implementation of forward-in-time population genetics simulations by abstracting basic operations required for simulating custom models. It provides generic procedures for *random sampling*, *mutation*, *recombination*, and *migration*.

SLiM (Messer 2013) is an efficient forward population genetic simulation designed for studying the effects of linkage and selection on a chromosome-wide scale. The program is aimed to simulate complex scenarios of demography and population substructure, various models for selection and dominance, realistic gene structure, and user-defined recombination maps. SLiM 2 (Haller and Messer 2017) is an evolutionary simulation framework that combines a powerful, fast engine for forward population genetic simulations with the capability of modeling a wide variety of complex evolutionary scenarios. One of the main features of SLiM 2 is scriptability which allows most aspects of the simulation to be tailored and customized. Furthermore, SLiM 2 have a graphical user interface called SLiMgui which allows an interactive scripting experience that makes development and testing of simulations vastly easier.

In this paper, we compare the performance achieved by our proposal to the performance reported by the Fwdpp (Thornton 2014) and the SLiM 2 (Haller and Messer 2017) software which showed good performance in terms of running time and memory storage.

2.2 Reference-based Genome Compression Approaches

Even the smallest genome sequences are large enough for computer representation, therefore, significant memory space is required for their storage. To tackle this problem, compression techniques are used to represent genome sequences (or any kind of data) using less memory size.

Chern et al. (2012) proposed an encoding algorithm composed of two phases: 1) generate a mapping from the reference genome to the target genome (based on the LZ77 algorithm), and 2) the mapping is compressed losslessly using an entropy coder. The decoder takes the compressed representation of the mapping and decompresses it by inverting the original mapping.

Wandelt and Leser (2013) presented a general open-source framework aimed to compress large-scale biological sequence data sets called Framework for REferential Sequence COmpression (FRESCO). FRESCO implements three approaches for storing reference entries: a) *standard LZ77-based* - to encode a sequence into a set of <match, length> pairs, b) *optimized LZ77-based* - similar to the previous but stores the original text when the length of the match is too small, and c) to encode each match into a reference as a triple <match, length, ncfm> (ncfm, next character following the match). The compression algorithm consists of matching prefixes of the target with sub-strings of the reference using a compressed suffix tree on reference. Furthermore, authors discuss three methods to increase compression ratios: *selection of a good reference*, *rewriting a reference*, and *second-order compression*.

The *Genotype Query Tools* (GQT) is proposed in (Layer et al. 2015) as a new indexing strategy and powerful toolset that enables interactive analyses based on genotypes, phenotypes and sample relationships.

An algorithm called *ERGC* (Efficient Referential Genome Compression), is presented in (Saha and Rajasekaran 2015). The ERGC algorithm is based on a reference genome and it is composed by the following stages: 1) to divide the reference and the target into equal-sized partitions, 2) to align each corresponding partition by using their algorithm based on hashing, and 3) to compress the starting positions and matching length using delta encoding. A new compression algorithm for collections of RNA-seq reads is presented in (Kingsford and Patro 2015). In (Nicolae, Pathak, and Rajasekaran 2015), the authors presented a lossless non-reference based FASTQ compression algorithm called LFQC (Lossless FASTQ Compressor). Zhang et al. (2015) proposed a lossless reference-based method namely FQZip for the compression of NGS data in FASTQ format. Later the authors presented a lossless light-weight reference-based compression algorithm namely LW-FQZip to compress FASTQ data (Zhang et al. 2015).

An algorithm called NRGC (Novel Referential Genome Compressor) is presented in (Saha and Rajasekaran 2016). The algorithm is based on a user-defined reference genome. In (Shi, Zhu, and Samsudin 2016) is proposed a reference-based compression method RDC for genome data in FASTQ format.

A general graph data structure is presented in (Ruths and Nakhleh 2013). The proposed data structure is devised for compressing the genotype space explored during a simulation run, along with efficient algorithms for constructing and updating compressed genotypes which support both mutation and recombination.

Good reviews on compression methods specialized for genome and reads compression and comparison of high-throughput sequencing data compression tools can be found in (Sardaraz, Tahir, and Ikram 2016), (Deorowicz and Grabowski 2013), (Numanagic et al. 2016), (Zhu et al. 2015).

3 GENETIC DRIFT LIBRARY

The `Libgdrift`(https://github.com/robertosolargallardo/libgdrift) simulation kernel is a `C++` library developed to simulate forward-in-time population genetics based on the Wright-Fisher model. `Libgdrift` provides a set of basic operations that may be combined to model and simulate arbitrary complex demographics scenarios. The summary of basic operations provided by `Libgdrift` is described as follows: **create**; to generate an initial fixed-size population, **split**; to emulate the partitioning of a population into different isolated populations (*genetic divergence*), **merge**; to combine a set of populations into a new unique population, **decrease** and **increase**; to emulate the variation in the population size (a decrease operation followed by an increase operation replicate a *bottleneck effect*), **migration**; to emulate the physical movement by individuals from one area to another (*founder effect*); and **extinction**; to emulate the death of all individuals of a population. A scenario consists of a set of basic operations (events) applied to populations organized as a *sankey diagram*, as shown in Figure 1, where nodes -colored rectangles- represent events and links -gray areas- represent populations. At the bottom of the figure, the values $t0 \dots t3$ represent the time the event should have occurred. A sankey diagram is a specific type of flow diagram in which the width of the links is shown proportionally to the flow quantity.

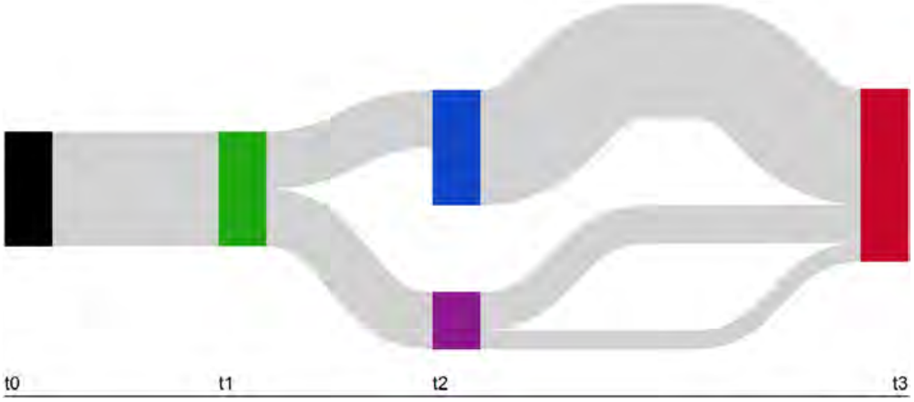


Figure 1: Sankey diagram.

The individuals of a population are characterized by their **ploidy** (number of sets of chromosomes), and a set of **chromosomes** that are composed by a set of **genes**. Genes may be either *short tandem repeat (STR)* (also known as microsatellite); tandem repeats of short DNA motifs (between 2 and 5 *base pairs* that are repeated several times), or *single-nucleotide polymorphism (SNP)*; variation in a single nucleotide that occurs at a specific position in the gene. We choose these types of genetic markers since are the most frequently used for generating genetic datasets in population genetics simulations. A genetic recombination occurs at gene level (intra-locus recombination is not yet supported). Furthermore, `Libgdrift` supports several mutation models: *JC69*, *K80*, *F81*, *HKY85* and *TN93* for SNP, and *stepwise mutation model* for STR. The `Libgdrift` library parses an input *simulation specification document* written in *JSON* to then run the simulation. Summary statistics over samples of populations are computed. Typically, summary statistics computed over samples correspond to diversity indices, such as: number of distinct haplotypes, number of segregating sites, mean and variance of pairwise differences, Tajima's D statistics, mean and variance of the numbers of the rarest nucleotide at segregating sites, number of private segregating sites, etc.

3.1 Performance Code Optimizations

`Libgdrift` aims to perform *as-fast-as-possible* forward-in-time simulations by implementing performance improvement code optimizations, such as: a) **recycling and reusing allocated memory**; to reduce the memory allocation overhead, b) **using principles of spatial and temporal locality of reference**; to amortize the high cost of memory accesses (cache-friendly code), and c) **two-phase gene compression**; to reduce the overall usage of main memory.

3.1.1 Cache-friendly Code

A cache-friendly code is a concept tightly coupled with the principle of locality. The cache-friendly code attempts to keep accesses contiguously allocated into memory so that you minimize cache misses (at CPU level). Caching is one of the main methods to reduce the impact of latency (CPU idle time). We adapt our data structures and temporal order of computations in order to maximize the use of the cache. All internal `Libgdrift` objects are stored into *array-like* structures (`std::vector` or native `C++` arrays), and are contiguously allocated into memory during the execution of the simulation. Furthermore, we adapt the main simulation cycle in order to **avoid unpredictable branches** and, as a consequence, the algorithmic complexity is significantly reduced (as shown in Section 3.1.3).

3.1.2 Recycling and Reusing Allocated Memory

A disadvantage of forward-in-time simulations is the way populations are handled during generation to generation transitions. Classical approaches create copies of each population to perform each transition. This is highly expensive in terms of memory allocation and it is even worst as the population size increases.

In this work, we propose to maintain two copies of each simulated population during the simulation: a source population pop_{src} and a destination population pop_{dst}. While random sampling is performed over pop_{src}, offspring individuals are stored into pop_{dst}. At the end of the generation and right before the next generation, populations pop_{src} and pop_{dst} are swapped. This code tweak helps us to reduce the number of memory allocations from $N*G$ to $2N$ - where N is the population size and G is the number of generations - thus, our method is independent of the amount of generations. The size of a population pop_{src} and pop_{dst} is increased iff the *increase* event is triggered.

3.1.3 Our Point Mutations Approach

A point mutation is a random change in one or a few base pairs in a SNP. Point mutations occurs during DNA replication and may involve a single base pair substitution, insertion or deletion. In forward-in-time simulations, the most compute-intensive section is the *main simulation cycle*, in which the transition between generations is performed while mutations per gene are evaluated by generating a random number per nucleotide, as described in Algorithm 1.

In this work, we propose to intervene the main simulation cycle in order to avoid unpredictable branches. Our point mutation approach consists of performing point mutations at the end of each generation instead of evaluating each gene per individual. To this end, we draw a random number from a binomial distribution $B_i = binomial(n, p)$, with $n = N*L(g_i)$ and $p = \mu_i$, where $L(g_i)$ is the number of nucleotides of gene g_i, N is the population size and μ_i is the mutation rate of gene g_i. The value of B_i indicates the total number of point mutations to be performed at the gene g_i, as shown in Algorithm 2. We use a binomial distribution because it models the possible number of times that a particular event can occur in a sequence of observations. Particularly, by using this approach allows us to reduce significantly the algorithmic complexity of the main simulation cycle from $O(N*nG*L(g_i))$ to $O(N)+O(B_i*nG*L(g_i))$, where nG is the number of

Algorithm 1 Main simulation cycle.

```
for $j = 0$ to $N$ do
  individual = $pop_{src}$[random[0,$N$)]
  for $i = 0$ to $nG$ do
    for $k = 0$ to $L(g_i)$ do
      if random[0,1] < $\mu_i$ then
        mutate(individual.$g_i[k]$)
      end if
    end for
  end for
  $pop_{dst}[j]$ = individual
end for
```

genes and taking into account that $B_i \lll N$.

Algorithm 2 Improvement to the main simulation cycle.

```
for $j = 0$ to $N$ do
  $pop_{dst}[j]$ = $pop_{src}$[random[0,$N$)]
end for
for $i = 0$ to $nG$ do
  $B_i$ = binomial$(N * L(g_i), \mu_i)$
  for $j = 0$ to $B_i$ do
    mutate($pop_{dst}$[random[0,$N$)]).$g_i$[random[0,$L(g_i)$)]
  end for
end for
```

4 TWO-PHASE COMPRESSION APPROACH

One of the main features of `Libgdrift` is to include a two-phase compression approach: **first-phase**; quaternary base conversion, and **second-phase**; reference-based gene compression.

4.1 Quaternary Base Conversion

Quaternary base conversion is a simple mechanism that consists of translating the four abbreviated DNA nucleotides in alphabetical order by using a simple function for mapping them into quaternary digits in numerical order as shown in Table 1. Since there are only four digits they can be represented by two binary digits, this way we can store four nucleotides per byte. Examples of this conversion are shown in Table 2. Thus, by using quaternary base conversion we reduce the memory usage approximately by 75% per sequence.

Table 1: Quaternary conversion of DNA nucleotides into base-4 digits.

Nucleotide	Mapped value	Bit representation
A	0	00
C	1	01
G	2	10
T	3	11

Table 2: Example of a quaternary base conversion.

Nucleotides	Quaternary	Binary	Byte
---C	0001	00000001	1
AGTA	0230	00101100	44
CGAT	1203	01100011	99
AGAT	0203	00100011	35
AGAG	0202	00100010	34
GGCA	2210	10100100	164

4.2 Reference-based Gene Compression

Reference-based compression consists of compressing a target genome given a known reference genome. The simulation of population genetics is focused on generating artificial genetic material of individuals of the same specie. Therefore, although the total size of the genome may be very large, each individual has similar genome with small variations (mutations). In this work, we propose a reference-based gene compression for simulation of large-scale population genetics scenarios. To this end, at the beginning of the simulation `Libgdrift` builds a *GenePool*, a tree-based data structure in which all genes and their variants are stored. Each variant points to a random generated gene of reference. Since individuals share most of their genetic material, they do not explicitly store their genes as attributes (no copies of existing instances), instead individuals point to *GenePool* entries.

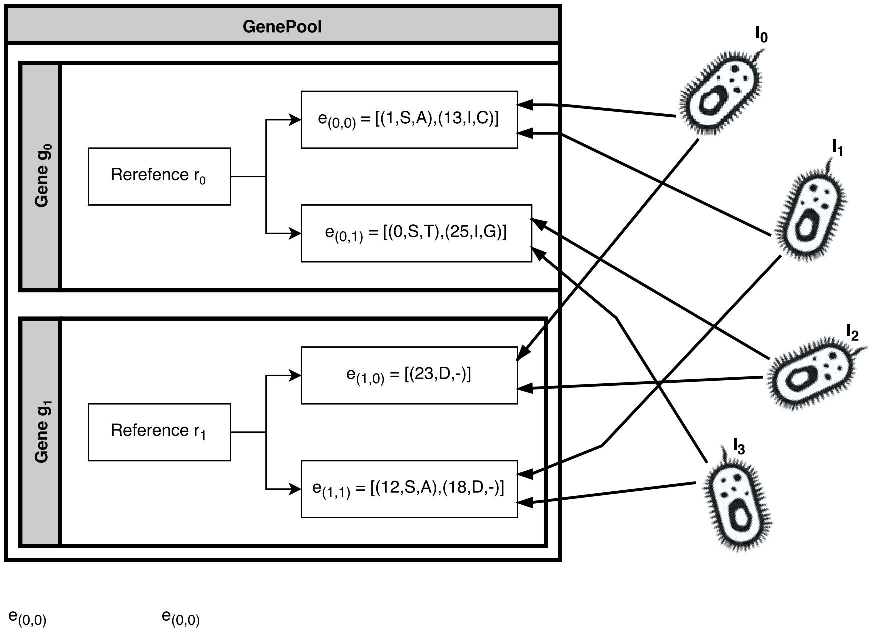


Figure 2: Example of reference-based gene compression: It can be observed that *GenePool* holds two genes (namely g_0 and g_1). Gene g_0 has two variants denoted as $e_{(0,0)}$ and $e_{(0,1)}$ pointing to their gene of reference r_0. The variant $e_{(0,0)}$ of gene g_0 indicates that individuals I_0 and I_1 have been affected by a nucleotide substitution at position 1 by an A, and an insertion of a new nucleotide C at position 13. Whereas gene g_1 also has two variants, where $e_{(1,1)}$ points the fact that a mutation of substitution of the nucleotide at position 12 occurred changing its value to A, and also a mutation deletion happened removing the nucleotide at position 18.

When a mutation is triggered at a $e_{(i,j)}$ (variant of gene g_i), a new *GenePool* variant $e_{(i,k)}$ ($k > j$) is created (or reused from the recycling bin). Each variant $e_{(i,k)}$ has a pointer to the gene of reference r_i, an internal control counter which indicates the number of individuals pointing it, and a list of mutations $< p,\ t,\ bp >$, where p is the position where the mutation is performed, t is the type of point mutation (substitution, insertion or deletion), and bp the new base pair (- when a deletion occurs). At the beginning of each generation all control counters are decreased by 1. When an individual points to an entry, its counter is increased by 1. At the end of the simulation all entries with counter equal to 0 are stored into a fixed-size recycling bin for further utilization.

5 EXPERIMENTAL RESULTS

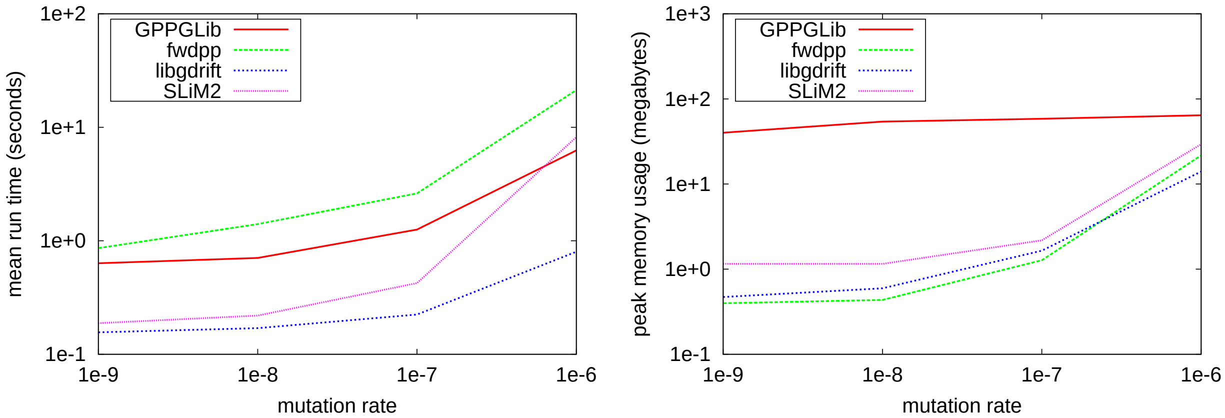
In this section we present the evaluation of the `Libgdrift` and compare its results to other state of the art alternatives. All of the C++ code were compiled using gcc version 5.3.1. Peak memory usage was measured by using ”*massif*”, a memory profiler which is part of the valgrind tools. Experiments were executed in the *Leftraru Cluster* from the National Laboratory for High Performance Computing (http://usuarios.nlhpc.cl/), Chile (NLHPC). The cluster is composed of four HP ProLiant SL250s Gen8 fat nodes and 128 HP ProLiant SL230s Gen8 thin nodes, all of them running Red Hat Enterprise Linux Server release 7.3. Nodes are interconnected by an Infiniband FDR 4X Mellanox 56Gbps network. Details about the nodes hardware is detailed in Table 3.

Table 3: Nodes hardware.

Node Model	Processor	Qty	Cores per Processor	RAM
HP ProLiant SL230s Gen8	Intel Xeon E5-2660	2	10	48GB
HP ProLiant SL250s Gen8	Intel Xeon E5-2660	2	10	64GB

The simulation parameters used for the experiments correspond to: i) population size, ii) mutation rate and iii) locus length. The values for the aforementioned parameters used in the experiments correspond to a population size of 1000 individuals, 1000 generations, mutation rates with values 1e−6, 1e−7, 1e−8 and 1e−9, and locus lengths of 1MB, 10MB and 1000MB. The performance metrics were obtained from the average of 100 executions. Execution times were obtained by taking measuring the code using `std::chrono` from C++11 at milliseconds level. For that purpose, the code was instrumented to take time measures on the isolated code segment that corresponds to the part where simulation cycles occur. This procedure was performed to the code of each analyzed simulator.

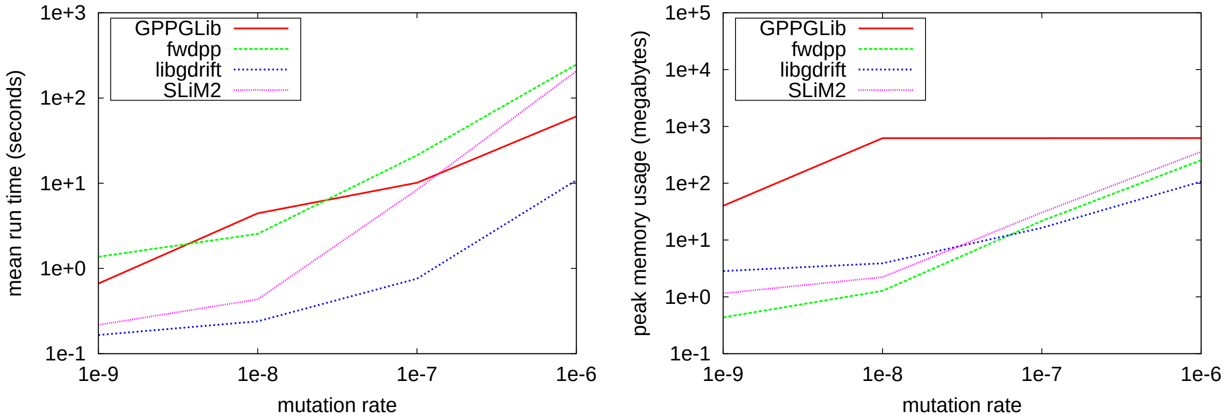
On one hand, Figure 3(a) shows the mean run time obtained for all simulators using a locus of length of 1 Megabyte. We observe that the mean run time of `Libgdrift` is lower than the other simulators for all cases. Specifically, is 20.423%, 29.246% and 89.548% faster than `Slim2` when using mutation rates of 1e−9, 1e−8 and 1e−7 respectively. The best performance in terms of run time is achieved by `Libgdrift` and occurs for a mutation rate of 1e−6, where our proposal is 6.802 times faster than `GPPGLib`. This advantage is taken from the reduction of the algorithmic complexity of the simulation cycle since the evaluation of mutations is performed at the end of each generation. Moreover, due to this we may appreciate that as the mutation rate is increased the advantage between `Libgdrift` and the rest is increased as well. On the other hand, Figure 3(b) shows the peak of memory usage obtained for all simulators using a locus of length of 1 Megabyte. We observe that `Libgdrift` and `fwdpp` have similar behavior in terms of memory usage. For mutation rates of 1e−9, 1e−8 and 1e−7, `fwdpp` uses 18.5674% 37.1215% and 29.3841% less memory respectively, but using a high mutation rate (1e−6), `Libgdrift` show a gain of 54.8879% (note that all graphics are in logscale in both axis). This advantage shows the effectiveness of our reference-based gene compression approach.



(a) mean run time (seconds)

(b) peak memory usage (megabytes)

Figure 3: Locus Length 1 MegaByte.

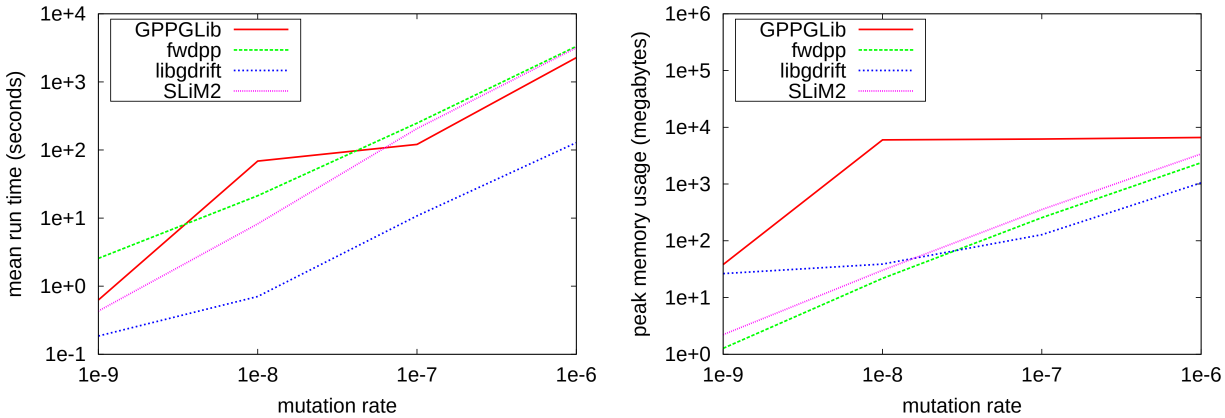


(a) mean run time (seconds)

(b) peak memory usage (megabytes)

Figure 4: Locus Length 10 MegaByte.

On one hand, Figure 4(a) shows the mean run time obtained for all simulators using a locus of length of 10 Megabyte. We observe a similar behavior in comparison with the previous case (locus of length of 1 Megabyte). `Libgdrift` overcomes by 31.4051%, 80.9903% and 1004.63% to `Slim2` for mutation rates of 1e−9, 1e−8 and 1e−7 respectively, and by an 460.617% to `GPPGLib` for a mutation rate of 1e−6. Furthermore, we may appreciate that as the locus length is increase, the gain in terms of acceleration is increased as well, although that the gap between `Libgdrift` and `GPPGLib` is reduced when we use a mutation rate of 1e−6. On the other hand, Figure 4(b) shows the peak of memory usage obtained for all simulators using a locus of length of 10 Megabyte. In this case, `fwdpp` uses 553.515% and 206.011% less memory than `Libgdrift` for mutation rates of 1e−9 and 1e−8 respectively. This behavior is explained by the fact that our proposal explicitly stores the sequence of reference to be able to apply mutation models, whereas other alternatives presented here do not explicitly store the DNA sequence. Nevertheless, `Libgdrift` uses a quaternary base conversion approach for nucleotide representation allowing us to use only two bits to store one nucleotide, that is, it requires just $0.25 * \text{locus\_length}$ bytes to store the sequence of reference into memory.Nevertheless, for high mutation rates; 1e−7 and 1e−6, `Libgdrift` overcomes



(a) mean run time (seconds)

(b) peak memory usage (megabytes)

Figure 5: Locus Length 100 MegaByte.

Table 4: Perf (http://web.eece.maine.edu/vweaver/projects/perf_events) statistics with locus length of 100 MB.

Library	LLC load misses	L1 dcache load misses
SLiM2	2E+11	2,9E+11
fwdpp	2E+10	4,3E+11
GPPGLib	3E+10	9,5E+10
libgdrift	1E+09	1,1E+10

to `fwdpp` for 32.8064% and 138.192% respectively. These results demonstrate us that `Libgdrift` uses an better approach for managing mutations.

The last experiment was executed using a locus of length of 100 Megabyte. Figure 5(a) shows the mean run time obtained for all simulators. We appreciate a similar trend to both previous cases but the gap between the `Libgdrift` curve and the rest is increased for high mutation rates. In this case, `Libgdrift` is 132.494% and 1065.2% faster than `Slim2` when we use low mutation rates; $1e-9$ and $1e-8$ respectively, and 1021.34% and 1667.06% faster than `GPPGLib` when we use high mutation rates; $1e-7$ and $1e-6$ respectively. This last experiment demonstrates that `Libgdrift` increases its performance with large-scale workloads in comparison with the rest of simulators. Figure 5(b) shows the peak of memory usage obtained for all simulators. On one hand, we observe that `fwdpp` shows a good performance in terms of memory usage for low mutation rates; $1e-9$ and $1e-8$ with 1973.24% and 78.8292% of advantage respectively (because we store the sequence of reference). On the other hand, `Libgdrift` shows a gain in term of memory usage in comparison with `fwdpp` of 99.4402% and 125.778% for high mutation rates; $1e-7$ and $1e-6$.

Finally, in Table 4 we show the statistics related to the last level cache (LLC) load misses and the L1 data cache load misses. In all cases, our proposal reports fewer misses than the other libraries. In particular, for the LLC our proposal reports 0,6% of the total misses reported by the SLiM2, and for the L1 the `Libgdrift` reports 2% of the total misses reported by the `fwdpp`.

6 CONCLUSIONS

In this paper, we presented a new library named `Libgdrift` for forward-in-time population genetics simulations. Our proposal is devised to improve the performance of the simulations. In other words, we aim to reduce the running time and at the same time the memory storage required to run the simulations. The `Libgdrift` library is based on a two-phase compression technique which allows to store four nucleotides per byte, which helps to significantly reduce the amount of memory required during the execution of each simulation.

To evaluate our proposal, we simulated the Wright-Fisher model. Results show that the `Libgdrift` simulation library can dramatically reduce the running time and the memory storage required by other well-known simulation software. Our contribution consists in a successful acceleration of forward-in-time simulation algorithms.

As future work we plan to include a three-phase compressing approach with second order compression (by compressing the gene of reference), and to study the effect of making effective mutations when the number of accumulated mutations overcomes a predefined threshold. Also, we will evaluate other models to study the flexibility of our library and implement the algorithm in a GPU based and/or Xeon Phi co-processor version.

ACKNOWLEDGMENT

This research was supported by the supercomputing infrastructure of the NLHPC Chile, partially funded by CONICYT Basal funds FB0001 and Fondef ID15I10560.

REFERENCES

Aberer, A. J., and A. Stamatakis. 2013. “Rapid Forward-in-Time Simulation at the Chromosome and Genome Level”. *BMC Bioinformatics* 14:216–216.

Cheng, H., D. Garrick, and R. Fernando. 2015. “XSim: Simulation of Descendants from Ancestors with Sequence Data”. *G3: Genes, Genomes, Genetics* 5 (7): 1415–1417.

Chern, B., I. Ochoa, A. Manolakos, A. No, K. Venkat, and T. Weissman. 2012. “Reference Based Genome Compression”. *CoRR* abs/1204.1912.

Deorowicz, S., and S. Grabowski. 2013. “Data Compression for Sequencing Data”. *Algorithms for Molecular Biology* 8 (1): 25.

Haller, B. C., and P. W. Messer. 2017. “SLiM 2: Flexible, Interactive Forward Genetic Simulations”. *Molecular Biology and Evolution* 34 (1): 230.

Kessner, D., and J. Novembre. 2014. “Forqs: Forward-in-Time Simulation of Recombination, Quantitative Traits and Selection”. *Bioinformatics* 30 (4): 576.

Kingman, J. 1982. “The Coalescent”. *Stochastic Processes and their Applications* 13 (3): 235 – 248.

Kingsford, C., and R. Patro. 2015. “Reference-Based Compression of Short-Read Sequences using Path Encoding”. *Bioinformatics* 31 (12): 1920.

Layer, R. M., N. Kindlon, K. J. Karczewski, E. A. C. ExAC, and A. R. Quinlan. 2015. “Efficient Compression and Analysis of Large Genetic Variation Datasets”. *bioRxiv*.

Messer, P. W. 2013. “SLiM: Simulating Evolution with Selection and Linkage”. *Genetics* 194 (4): 1037–1039.

Nicolae, M., S. Pathak, and S. Rajasekaran. 2015. “LFQC: A Lossless Compression Algorithm for FASTQ Files”. *Bioinformatics* 31 (20): 3276.

Numanagic, I., J. K. Bonfield, F. Hach, J. Voges, J. Ostermann, C. Alberti, M. Mattavelli, and S. C. Sahinalp. 2016, October. “Comparison of High-Throughput Sequencing Data Compression Tools”. *Nature Methods* 13 (12): 1005–1008.

Ruths, T., and L. Nakhleh. 2013. “Boosting Forward-Time Population Genetic Simulators Through Genotype Compression”. *BMC Bioinformatics* 14 (1): 192.

Saha, S., and S. Rajasekaran. 2015. “ERGC: An Efficient Referential Genome Compression Algorithm”. *Bioinformatics* 31 (21): 3468.

Saha, S., and S. Rajasekaran. 2016. "NRGC: A Novel Referential Genome Compression Algorithm". *Bioinformatics* 32 (22): 3405.

Sardaraz, M., M. Tahir, and A. Ikram. 2016. "Advances in High Throughput DNA Sequence Data Compression". *Journal of Bioinformatics and Computational Biology* 14 (03): 1630002.

Shi, H., Y. Zhu, and J. Samsudin. 2016. "Reference-Based Data Compression for Genome in Cloud". In *ICCIP*, 55–59.

Tahmasbi, R., and M. C. Keller. 2017. "GeneEvolve: A Fast and Memory Efficient Forward-Time Simulator of Realistic Whole-Genome Sequence and SNP Data". *Bioinformatics* 33 (2): 294.

Thornton, K. R. 2014. "A C++ Template Library for Efficient Forward-Time Population Genetic Simulation of Large Populations". *Genetics* 198 (1): 157–166.

Wandelt, S., and U. Leser. 2013, September. "FRESCO: Referential Compression of Highly Similar Sequences". *IEEE/ACM Trans. Comput. Biol. Bioinformatics* 10 (5): 1275–1288.

Yang, X., X. Ni, Y. Zhou, W. Guo, K. Yuan, and S. Xu. 2016. "AdmixSim: A Forward-Time Simulator for Various and Complex Scenarios of Population Admixture". *bioRxiv*.

Yuan, X., D. J. Miller, J. Zhang, D. Herrington, and Y. Wang. 2012. "An Overview of Population Genetic Data Simulation". *Journal of Computational Biology* 19 (1): 4254.

Zhang, Y., L. Li, J. Xiao, Y. Yang, and Z. Zhu. 2015. *FQZip: Lossless Reference-Based Compression of Next Generation Sequencing Data in FASTQ Format*, 127–135. Cham: Springer International Publishing.

Zhang, Y., L. Li, Y. Yang, X. Yang, S. He, and Z. Zhu. 2015, Jun. "Light-Weight Reference-Based Compression of FASTQ Data". *BMC Bioinformatics* 16 (1): 188.

Zhu, Z., Y. Zhang, Z. Ji, S. He, and X. Yang. 2015. "High-Throughput DNA Sequence Data Compression". *Briefings in Bioinformatics* 16 (1): 1.

AUTHOR BIOGRAPHIES

ROBERTO SOLAR is an postdoctoral researcher at CITIAPS (Centre for Innovation in Information Technologies for Social Aplicactions), Universidad de Santiago de Chile, Chile. He holds a Ph.D. in High Performance Computing from Universitat Autonoma de Barcelona, Spain. His e-mail address is roberto.solar@usach.cl.

VICTOR SEPULVEDA is a research and development engineer at CeBiB (Centre for Biotechnology and Bioengineering), Chile. He has a M.Sc. in computer science from Universidad de Chile. His email address is vsepulve@dcc.uchile.cl.

ALONSO INOSTROSA-PSIJAS is a postdoctoral researcher at CITIAPS (Centre for Innovation in Information Technologies for Social Aplicactions), Universidad de Santiago, Chile. He holds a Ph.D. Informatics Engineering from Universidad de Santiago de Chile. His email address is alonso.inostrosa@usach.cl.

VERONICA GIL-COSTA received her PhD (2009) in Computer Science, from Universidad Nacional de San Luis (UNSL), Argentina. She is a former researcher at Yahoo! Labs Santiago hosted by the University of Chile. She is currently an assistant professor at the University of San Luis and researcher at the National Research Council (CONICET) of Argentina. Her email address is gvcosta@unsl.edu.ar.

MAURICIO MARIN is a former researcher at Yahoo! Labs Santiago hosted by the University of Chile, and currently a full professor at University of Santiago, Chile. He holds a PhD in Computer Science from University of Oxford, UK, and a MSc from University of Chile. His email address is mauricio.marin@usach.cl.