

Large-Scale Simulations of Bacterial Populations Over Complex Networks

ANDREIA SOFIA TEIXEIRA¹, PEDRO T. MONTEIRO³, JOÃO A. CARRIÇO²,
FRANCISCO C. SANTOS¹ and ALEXANDRE P. FRANCISCO³

ABSTRACT

The understanding of bacterial population genetics and evolution is crucial in epidemic outbreak studies and pathogen surveillance. However, all epidemiological studies are limited to their sampling capacities which, by being usually biased or limited due to economic constraints, can hamper the real knowledge of the bacterial population structure of a given species. To this end, mathematical models and large-scale simulations can provide a quantitative analytical framework that can be used to assess how or if limited sampling can infer the true population structure. In this article, we address the large-scale simulation of genetic evolution of bacterial populations, using Wright–Fisher model, in the presence of complex host contact networks. We present an efficient approach for large-scale simulations over complex host contact networks, using MapReduce on top of Apache Spark and GraphX API. We evaluate the relation between cluster computing power and simulations speedup and include insights on how bacterial population diversity can be affected by mutation and recombination rates, and network topology.

Keywords: GraphX, large-scale simulations, graph-parallel computations, spark, population genetics.

1. INTRODUCTION

UNDERSTANDING A GIVEN BACTERIAL SPECIES POPULATION STRUCTURE and how it is shaped by genetic forces of mutation and recombination is vital for interpreting the response of bacterial populations to selection pressures, such as antibiotic treatment or vaccination (Robinson et al., 2010). In this context, large-scale studies are fundamental not only to understand such processes, but also to validate both models and methods, such as phylogenetic inference algorithms, and to understand how the sampling that is commonly done can affect or bias our perception of the true population structure. However, it is often difficult to execute such large-scale studies with real pathogen samples due to economic and practical reasons. Moreover, most publicly available datasets are biased to their original studies. Hence, given the model complexity and required population sizes, large-scale simulations are the most convenient way to address this issue.

¹ATP Group, INESC-ID Lisboa and Instituto Superior Técnico, Universidade de Lisboa, Lisboa, Portugal.

²Faculdade de Medicina, Instituto de Microbiologia and Instituto de Medicina Molecular, Universidade de Lisboa, Lisboa, Portugal.

³INESC-ID Lisboa and Instituto Superior Técnico, Universidade de Lisboa, Lisboa, Portugal.

Previous studies have shown that observed population genetic structure of several important human pathogens, such as *Streptococcus pneumoniae* and *Neisseria meningitidis*, can be explained using a simple evolutionary model (Fraser et al., 2005, 2007, 2009; Hanage et al., 2006). This model was based on neutral mutational drift and incorporated recombination events, but was tested only for panmictic populations, that is, all the individuals in the model could freely exchange DNA through recombination events. Although this simple evolutionary model works well for local populations, at a “microepidemic” level, its predictions no longer fit observed genetic relationships of large and widely distributed bacterial populations. With the increasing volume of data obtained with sequence-based typing methods, such as multilocus sequence typing (MLST) (Maiden et al., 1998) (currently the gold standard for epidemiological surveillance for many bacterial species), a much more complex pattern emerges that cannot be explained solely by the simple “microepidemic” assumption.

The evolution of transmissible bacteria occurs by mutation and recombination, and is influenced by epidemiological as well as molecular processes. These aspects are fundamental in the process of strain diversification (Spratt et al., 2001), and as a mechanism by which strains acquire virulence factors or resistance determinants (Ochman et al., 2000). On the other hand, microbial evolution is also influenced by the environment and by host contact networks, which modulates the spread of microbial pathogens. The study of the impact of host contact network topologies, and associated transmission ratios, on bacterial population evolution and genetic diversity becomes then relevant, increasing the model complexity.

Large-scale simulations are, however, computationally demanding, in particular when model complexity increases. In this work, we consider an extension of the above simple evolutionary model by incorporating the underlying host contact network. We propose an extension of the simulation framework for large bacterial populations presented by Teixeira et al. (2017), which implements the Wrigh–Fisher model (Tran et al., 2013), on top of Apache Spark (Zaharia et al., 2010), making use of both MapReduce programming model and GraphX API (Xin et al., 2013). This extended version includes improvements to the code, an implementation of the Simpson’s Index of Diversity (SID) (Simpson, 1949) and a more realistic parameterization, which is used to analyze population diversity under different input parameters.

We center our discussion in two main aspects: how these large-scale simulations benefit from parallelization, evaluating inherent parallelism limits and drawing conclusions on the relation between cluster computing power and simulation speedup; and how bacterial populations are affected by network topology and inherent mutation and recombination rates. We used Google Cloud Platform to conduct our experiments and to analyze the performance of each simulation for each network topology.

The article has the following structure. The simulation and computational models are described in Section 2. In Section 3, we describe our implementation. Experimental evaluation results are discussed in Section 4, followed by the conclusions and future work in Section 5.

2. APPROACH

SparkNetSim is a parameterizable framework for simulating the evolution of large-scale bacterial populations over complex host contact networks. We focus on bacterial population genetics, where isolates are represented as typing profiles, which encodes for a specific genetic lineage. There are several typing methods and, in this work, we consider the MLST technique. In MLST, DNA sequences are obtained for a set of typically seven housekeeping loci, and different sequences identified at each locus are assigned as different alleles through a unique identifier (Maiden et al., 1998). Nowadays, it is common to find profiles with hundreds of loci.

In our simulations, each isolate is represented through a profile that may be subject to transformations along time, under the influence of genetic events, namely mutation and recombination, which can be in turn modulated by the environment. Let each strain in a bacterial population be then characterized by a profile, with a profile being defined by the combination of its alleles, a vector of labels. All profiles have the same length and, for each position, different labels among profiles mean different alleles.

In our simulator, the environment is represented by the host contact network. Given such a network, the bacterial population at each host evolves according to a neutral evolutionary model (Fraser et al., 2005), which, in turn, based on the neutral infinite alleles model (IAM) (Kimura, 1968). It assumes that genetic events do not contribute to the fitness of the individual and, therefore, all individuals in a population have an equal chance of reproducing and being subjected to genetic events. Under IAM, mutation always

generates a new allele, leading to new profiles, also known as sequence types (ST). Recombination, on the other hand, introduces an existing allele randomly selected from the isolates present in the previous generation, which may lead to novel allelic profiles, that is, ST, or to the reappearance of existing ones. Mutation and recombination occur independently, with each event being rare and mutation taking precedence over recombination. A new generation is obtained at each step, leading to nonoverlapping generations.

The interactions between hosts take place by allowing pathogens to migrate from one host to another. Migration occurs according to the contact network topology, the migration frequency, and the edge transmission probabilities are defined by the user. After each migration phase, the population at each host is obtained through sampling with replacement from the set of both the individuals already at the host and those that migrated to it.

We should note that the model based on IAM described above is also known as the neutral Wright–Fisher model (Tran et al., 2013), where equal fitness means that all individuals can be picked as a parent with the same probability.

Since we are interested in large-scale simulations, we rely on Apache Spark (Zaharia et al., 2010) and GraphX (Xin et al., 2013) to parallelize and scale up our simulations. Apache Spark extends the MapReduce model and enables the creation of iterative programs. MapReduce (Lin and Dyer, 2010) is a high-level programming model proposed to address embarrassingly parallel data processing problems (Dean and Ghemawat, 2008). The main ingredient of Apache Spark for improving scalability and maintaining fault tolerance is the use of resilient distributed datasets (RDD) (Zaharia et al., 2012). On the other hand, it provides new levels of abstraction over Apache Hadoop and high-level APIs, usable through several programming languages. GraphX API is one of those APIs designed to address graph-parallel problems (Xin et al., 2013).

The use of the MapReduce programming model for solving a problem requires redesigning algorithms around *Map* and *Reduce*. On the other hand, since we are exploiting the parallelism among mappers and reducers, running on top of a shared distributed file system, special care is required to avoid synchronization issues. Note that although MapReduce, and in our particular case Apache Spark, hides most of synchronization issues, wrongly designed algorithms may not be able to benefit from inherent parallelism.

It is then important to analyze some basic requirements, knowing that in the *Map* phase a function is applied to many key-value pairs in parallel, generating another set of key-value pairs, and that in the *Reduce* phase those intermediate key-value pairs are aggregated by key and a function is applied to each set of values, generating new key-value pairs. Returning to our simulation problem, we have two main tasks: (1) the evolution of each population at each node where mutation and recombination take place; and (2) the exchanges between nodes and the replacement of each population taking into account the samples of the populations migrated from the connected neighbors.

For the first task, although each host has its own population, evolving independently, we must take care of assigning unique global identifiers on mutation events, and sampling STs uniformly among each host population as allele donors on recombination events. We address the first issue by introducing a special schema for generating unique allele identifiers (see implementation for details). The second issue implies grouping each population for each host, a step that we cannot avoid and that will lead to some performance lost.

For the second task, each node has to receive a migrating sample of the current population of its neighbors, mix migrating individual with its own population, and create a new population through sampling. Besides being a simple problem when thinking about the MapReduce model, we can identify some similarities between this process and PageRank (Page et al., 1999) or Label Propagation (Zhu and Ghahramani, 2002) problems. Both problems rely on exchanging information among neighbors to update their state, and both have already been implemented using MapReduce and, in particular, making use of GraphX.

3. IMPLEMENTATION

Let $G=(V, E)$ be a connected and weighted graph, with $n=|V|$ nodes and $m=|E|$ edges, and with an edge transmission probability function $w: E \rightarrow \mathbb{R}$. Let t be the number of times that the sequence of number of evolutions followed by the number of exchanges happens.

The simulator (<https://bitbucket.org/steixeira/sparknetism>) takes several parameters: (1) the population size of each node p ; (2) the file containing the populations; (3) the file containing the network; (4) mutation rate per allele per generation, m ; (5) recombination rate per allele per generation, r ; (6) number of evolutions, $evols$; (7) number of exchanges, $exchs$; (8) number of times for the cycle $evols$ followed by $exchs$ to happen, t ; (9) frequency of sampling/writing on disk s ; (10) local or cluster mode; (11) number of partitions; (12) partition to be applied on the Graph; (13) directory to write the results. Each iteration of evolution is considered as one generation.

The workflow consists of two main steps: (i) sequence of evolutions for each population, followed by (ii) sequence of exchanges between nodes. This workflow is performed t times.

The first step consists of evolving each population according to the Wright–Fisher model. For each individual, an allele can be subject to both mutation and recombination events with probabilities m and r , respectively. In the second step, individuals migrate among host populations as follows: for each host u , given a neighbor v of u , u samples a given proportion of its population, according to the edge transmission probability $w(u, v)$, and sends it to v ; each host receives the population samples sent by its neighbors and creates a pool with those samples mixed with its own population; a new population is built, with the same size as the previous one, but where individuals are chosen randomly from the population pool obtained in the previous step. At the end of this process, all populations are persisted on secondary storage.

Given several samples along generations, we can then evaluate and validate several models and parameters. In this work we focus on diversity changes along time, leading to some exciting results since we are not explicitly considering selection mechanisms. It is possible to calculate the diversity with the SID, which is the probability that two individuals randomly selected from a sample will belong to the same species (Simpson, 1949).

Let us explain how simulations and the computation of SID can be implemented using Apache Spark and GraphX. This will allow us to conduct large-scale simulations and analyses.

For the simulations of bacterial populations we need two input files. The first input file contains the bacterial population for all hosts. This file has an individual per line represented as a *key/value* pair, where the *key* corresponds to its host/node identifier and the *value* corresponds to the sequence of its alleles. Although alleles are usually characterized by an integer in real datasets, we need a different characterization as discussed above since we must assign unique global identifiers on mutation events. Each individual is then characterized by an array of strings, where each string is an allele in the form $X.Y.Z$, where X corresponds to the identifier (ID) of the node/host, where the mutation occurs, Y is the generation id, and Z is the number corresponding to the Z -th mutation event of that generation in that node. With this change we guarantee the requirements of the IAM regarding allele uniqueness on mutation events.

The second input file is the network file which contains an edge list with transmission probabilities, that is, a triple per line with the source u , the destination v , and the fraction $w(u, v)$ of the population to be sent from the u to v . Note that, since graphs are assumed to be directed in GraphX, if we want an undirected network, then the edge list must contain edges in both directions. For the exchange process among neighbors, it is necessary for the *Map* phase that each node sends to itself its own population. Each node must have then an edge to itself with a probability of transmission 1.0. We load the input files with the *SparkContext.textFile* method, which maps the inputs into RDDs. This method allows the definition of a minimum number of partitions for the data that is received as a parameter and that must adapted according to our cluster configuration.

Evolution and exchanges among nodes can be implemented as independent *Map* and *Reduce* tasks. Listing 1 provides the pseudocode for the main routine. Full code is available at <https://bitbucket.org/steixeira/sparknetsim>.

After loading the input files into RDDs, we apply the *groupByKey* transformation in the RDD of the population to guarantee that each population is in the same data structure. This is a requirement to perform the Wright–Fisher model, as recombination demands elements from the same population to be recombined. This transformation, when called on a dataset of (K, V) pairs, returns a dataset of $(K, \text{Iterable}(V))$ pairs. Once we have each population gathered in the same data structure, we perform a *Map* transformation to make each population evolve in parallel. At the end of each iteration, we have to cache intermediate results and to apply *take* to guarantee that the new RDD contains the new population. We sample then the populations and we persist them using the operation *saveAsTextFile*.

For the exchange process, we build a *PropertyGraph* using the RDD of the population and the RDD

Listing 1: Evolutionary process.

```

val popData=sc . textFile (populations, npartitions) . cache ()
val netData=sc . textFile (network, npartitions) . cache ()
val individualsRDD=popData . map{* process input file *} . groupByKey () . cache ()

// Evolutions
val nextpop=individualsRDD . map{ i=>
  val idpop=i . _1
  val population=i . _2
  for (each individual in population) {
    // Recombintation | // Mutation
    population . update (i, individual)}
  (idpop, population)}
nextpop . map (* format output as desired *) . saveAsText File (output dir)
individualsRDD=nextpop

// Exchanges

val newpop=graphpopulation . triplets . map { t=>
  // generate a collection with the proportion
  // of population to send
  (dest, fractionofpopulation)
} . reduceByKey (- ++ - ) . map{ i=>
  //create the new populations by sampling
  (key, newpopulation)}
newpop . map (* format output as desired *) . saveAsTextFile (outputdir)
individualsRDD=newpop

```

with the edges from the host contact network. We also allow the user to define which *GraphPartition* (provided by GraphX) to use as one of the arguments. If the parameter is 0, then no *GraphPartition* is used; if it is 1, then we use the strategy *PartitionBy2D*.

Once the *PropertyGraph* has been created, we use the *triplets* view, which allows the access to both node and edge properties, that is, the populations and the fraction of each population to be sent, respectively. At each exchange step, we create the samples from each source node, represented as a new RDD with the key being the destiny node and the value being the sample. Another *reduceByKey* transformation with the concatenation operator (+ +) allows us to join all the samples with the same key, providing a collection with all pools of elements from which new populations will be sampled. Each new population is generated through sampling with a *Map* transformation. At the end of this process, we also call the operation *saveAsTextFile* over the RDD of the new populations.

For the calculation of the SID, we need an input file where each line is in the format of [*generation node individual*] (this is also the output format of our simulator). After loading data into an RDD and also defining in how many partitions we want to partition our data, we apply the *groupByKey* transformation to aggregate values for each node, per generation. We use then the *Map* transformation to process each node per generation in parallel. It is now straightforward to compute the SID using operators *groupBy(identity)*, which groups each unique individual, and then *mapValues* to count how many unique sequences there are.

We should note that when we run large-scale simulations, the output can easily reach dozens of gigabytes. In addition, the SID calculation with Apache Spark for an output size of 30 GB takes seconds. In the end, we use again *saveAsTextFile* to write on disk the SID of each node, for each generation.

4. RESULTS

We present in this section experimental results concerning both performance evaluation and population diversity growth evaluation. These results rely on homogeneous and heterogeneous host contact networks.

For homogeneous networks we used fully connected networks, representing well-mixed populations, and regular networks. In the latter, we consider regular ring lattices, where each node has the same number of neighbors and is connected to those that are closest (Watts and Strogatz, 1998). We consider scale-free

networks generated by the B-A model proposed by Barabási and Albert (1999). In the B-A model, networks are built through the combination of growth and preferential attachment: each time step one adds a new node with m edges that link it to m different nodes (growth). To choose the m nodes one assumes that the probability of linking to a node u scales linearly with the degree of u (preferential attachment) Barabási and Albert (1999). This algorithm creates a network with a power-law degree distribution and low clustering coefficient. To also test a scale-free network with high clustering, we adopt a minimal scale-free model proposed by Dorogovtsev et al. (2001). In this case, we also have growth, yet each new node attaches to both ends of a randomly chosen edge. As such, this rule favors the creation of triangular relations between individuals, thereby greatly enhancing the clustering coefficient of the final network, yet portraying the same power-law degree distribution as the B-A model. Except for the fully connected network (where all nodes have degree $n - 1$), all networks have average degree 4.

4.1. Performance evaluation

Apache Spark can run either on local or on cluster modes. We relied on both modes for our experiments to compare how running time scales, parameterizing simulations with different population sizes and different graphs. Experiments were conducted in a cluster hosted at Google Cloud Platform (<https://cloud.google.com>), configured with different number of workers: 2, 4, 8, and 16. Each worker is an Intel(R) Xeon(R) CPU @ 2.50 GHz with 4 cores and 16 GB of RAM. To achieve the best results possible, in each experiment we partitioned the input in as many partitions as the number of the cores available. The local setup was performed in a worker node, using only one core.

Given the evolution model coupled with a host contact network, leading to both evolutions and interactions between populations, the operations in local mode take considerable time, with each step being executed sequentially. Also, it scales reasonably well when in cluster mode. For a precise comparison, in what concerns scalability, the simulator parameters are fixed as follows: the size of the population per node is 1000, the mutation rate is 0.001, the recombination rate is 0.01, the number of evolutions per time step is 25, the number of exchanges per time step is 1, the number of time steps is 10, the frequency for writing on disk is 10 generations, and the transmission probability is 0.01 for all edges.

The first observation is that writing on disk is the most costly operation. This is even more noticeable in the new version of the simulator presented in this article, where data are persisted for each main iteration (and used afterward to compute for instance the SID). We address this issue by using the *saveAsTextFile* method available in Apache Spark, exploiting the ability to parallelize write operations on the underlying distributed file system through data partitioning.

We consider three different network topologies as described before: cliques, regular networks, and two scale-free networks (Barabási and Albert, 1999; Dorogovtsev et al., 2001) sharing the same degree distribution, yet portraying different clustering coefficients. The clique topology leads to the highest computational cost when performing exchanges. Scale-free networks are more realistic for host contact networks, but being very sparse lead to much less work during exchanges. Tests were run for fully built topologies, an average degree of 4, and different network sizes. The running time results averaged over 10 runs are presented in Table 1. We computed also the speedup given by the quotient between the time spent on one processor and the time spent on multiple processors, results of which are presented in Table 2 and in Figure 1. Moreover, we computed the efficiency given by the quotient between the speedup and the number of processors, results of which are presented in Table 3.

We can observe that the speedup becomes more evident as networks grow in size. Running in cluster mode, and increasing the number of workers and cores, result in a significant reduction in the running time, namely for larger networks. The fact that we can compute the population evolution at each node independently seems to be exploited as expected. The same happens with the exchange process among node populations.

When designing parallel algorithms, one of the analyses that should be done is to estimate the relation between achievable speedups and the number of workers/cores. Amdahl's law is useful in this context to both understand results and project expected speedups (Amdahl, 1967). The main points are that we should only optimize if the fraction that can be optimized constitutes a large portion of the overall time, and that if the optimization is effective, the obtained speedup is largely determined by the strictly sequential fraction. Although this fraction constituted only a small fraction of the initial time, since it cannot be optimized, it will represent a larger fraction as more parallelism is allowed. Given k workers (cores) and a program that

TABLE 1. RUNNING TIME IN SECONDS FOR DIFFERENT TOPOLOGIES AND NETWORK SIZES

Topology	Size (n)	Mode				
		Local 1 core	2 workers 8 cores	4 workers 16 cores	8 workers 32 cores	16 workers 64 cores
Clique	200	2466.4	225.0	218.2	93.8	89.4
	500	7315.4	852.6	446.2	228.4	152.8
	1000	16994.6	3827.8	1077.0	515.4	315.0
	2000	39427.5	13436.2	4906.6	1507.4	746.2
Regular	200	1487.8	149.2	157.4	73.8	69.0
	500	2271.2	437.2	385.2	137.6	97.4
	1000	4134.4	1576.6	862.6	258.6	163.6
	2000	6912.4	5037.6	1746.8	534.6	303.4
Barabási-Albert	200	1456.0	153.4	145.4	75.4	69.0
	500	2575.2	471.8	383.2	134.0	102.2
	1000	4342.2	1602.8	867.2	253.8	164.6
	2000	6564.2	5038.4	1801.4	549.0	305.2
Minimal	200	1448.2	150.0	142.6	75.6	67.2
	500	2829.4	442.2	391.8	135.8	92.6
	1000	4377.8	1600.0	850.6	248.0	163.8
	2000	6682.2	4692.0	1825.6	547.0	284.0

spends a fraction f of time on operations that are infinitely parallelizable, and the remaining fraction $1 - f$ on strictly sequential operations, the overall speedup is given by $1/((1 - f) + f/m)$. In Figure 1 we can observe that, as we increase the size of the clique networks, we obtain a higher speedup as we increase the number of workers.

According to Amdahl's law, we are observing $f > 99\%$. For the scale-free networks, because they are much sparser than cliques, with less work performed on exchange, we observe about $f = 97\%$ and speedups are small above 32 cores. This seems to point out that the simulator is highly parallelizable, independently of the network size, which is an important observation if simulations with much larger networks are desirable. Note also that the running time grows almost linearly as we increase the number of nodes (see Table 1). Although expected, this observation shows that the evolution of populations occurs independently

TABLE 2. SPEEDUP FOR DIFFERENT TOPOLOGIES AND NETWORK SIZES

Topology	Size (n)	Mode				
		Local 1 core	2 workers 8 cores	4 workers 16 cores	8 workers 32 cores	16 workers 64 cores
Clique	200	1.0	11.0	11.3	26.3	27.6
	500	1.0	8.6	16.4	32.0	47.9
	1000	1.0	4.4	15.8	33.0	54.0
	2000	1.0	2.9	8.0	26.2	52.8
Regular	200	1.0	9.7	10.1	19.2	21.6
	500	1.0	6.4	7.2	20.8	30.6
	1000	1.0	2.7	5.1	17.7	26.7
	2000	1.0	1.4	3.7	12.2	23.5
Barabási-Albert	200	1.0	10.0	9.5	20.3	21.6
	500	1.0	5.2	5.9	16.5	23.3
	1000	1.0	2.6	4.8	16.0	25.3
	2000	1.0	1.4	4.0	13.0	22.8
Minimal	200	1.0	9.5	10.0	19.3	21.1
	500	1.0	5.5	6.7	19.2	25.2
	1000	1.0	2.7	5.0	17.1	26.3
	2000	1.0	1.3	3.6	12.0	21.5

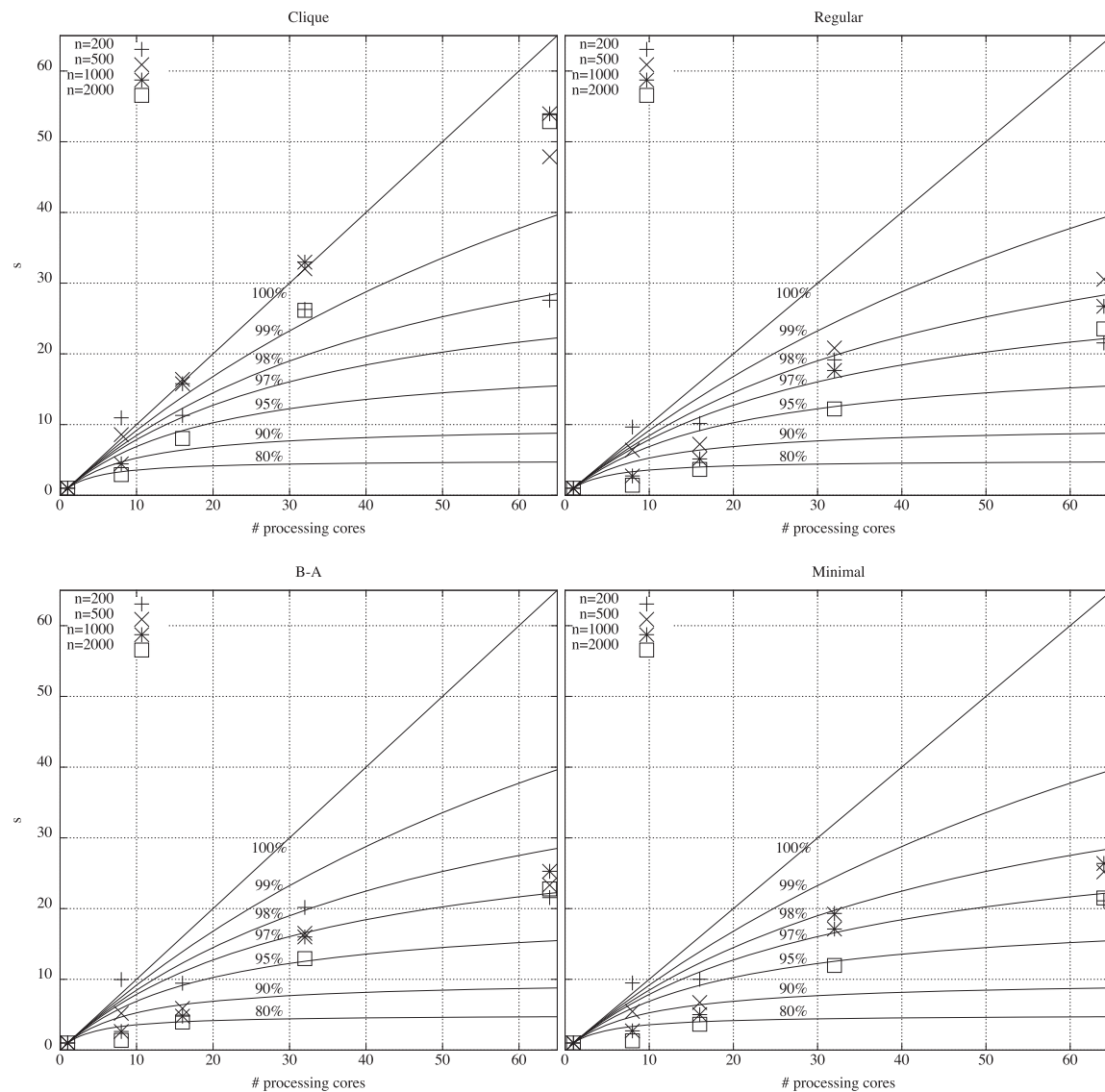


FIG. 1. Speedup as a function of the number of available cores for cliques, regular networks, Barabási-Albert (B-A) networks, and minimal scale-free networks, with different network sizes (n). The curves are provided by Amdahl's law (Amdahl, 1967), where the percentage corresponds to the fraction that is infinitely parallelizable.

in our implementation, where special care was taken in what concerns the parallelization of recombination and mutation events, while using the IAM model.

4.2. Population diversity evaluation

We now analyze the impact of the host contact network topology on the bacterial population genetic diversity for different rates of mutation and recombination. We are interested to see if strain persistence and local evolutionary events, reflecting local expansion, and limitation of genetic exchange due to the host contact network can lead to observable variations of SID values that can erroneously be attributed to selection events. For that, we run simulations for 2500 generations (with cycles of 25 evolutions followed by one exchange between nodes), and compute the SID for each node.

We used four network topologies of size 1000, as described before: a clique, a regular (ring) network, and two scale-free networks, one with low clustering coefficient [Barabási-Albert Model (Barabási and Albert, 1999)], and one portraying high clustering [Dorogovtsev-Mendes-Samukhin *minimal* scale-free model (Dorogovtsev et al., 2001)]. All networks (besides the clique) share an average degree of 4. The

TABLE 3. EFFICIENCY FOR DIFFERENT TOPOLOGIES AND NETWORK SIZES

Topology	Size (n)	Mode				
		Local 1 core	2 workers 8 cores	4 workers 16 cores	8 workers 32 cores	16 workers 64 cores
Clique	200	1.00	1.38	0.71	0.82	0.43
	500	1.00	1.08	1.03	1.00	0.75
	1000	1.00	0.55	0.99	1.03	0.84
	2000	1.00	0.36	0.50	0.82	0.83
Regular	200	1.00	1.21	0.63	0.60	0.34
	500	1.00	0.80	0.45	0.65	0.48
	1000	1.00	0.34	0.32	0.55	0.42
	2000	1.00	0.18	0.23	0.38	0.37
Barabási-Albert	200	1.00	1.25	0.59	0.63	0.34
	500	1.00	0.65	0.37	0.52	0.36
	1000	1.00	0.33	0.30	0.50	0.40
	2000	1.00	0.18	0.25	0.41	0.36
Minimal	200	1.00	1.19	0.63	0.60	0.33
	500	1.00	0.69	0.42	0.60	0.39
	1000	1.00	0.34	0.31	0.53	0.41
	2000	1.00	0.16	0.23	0.38	0.34

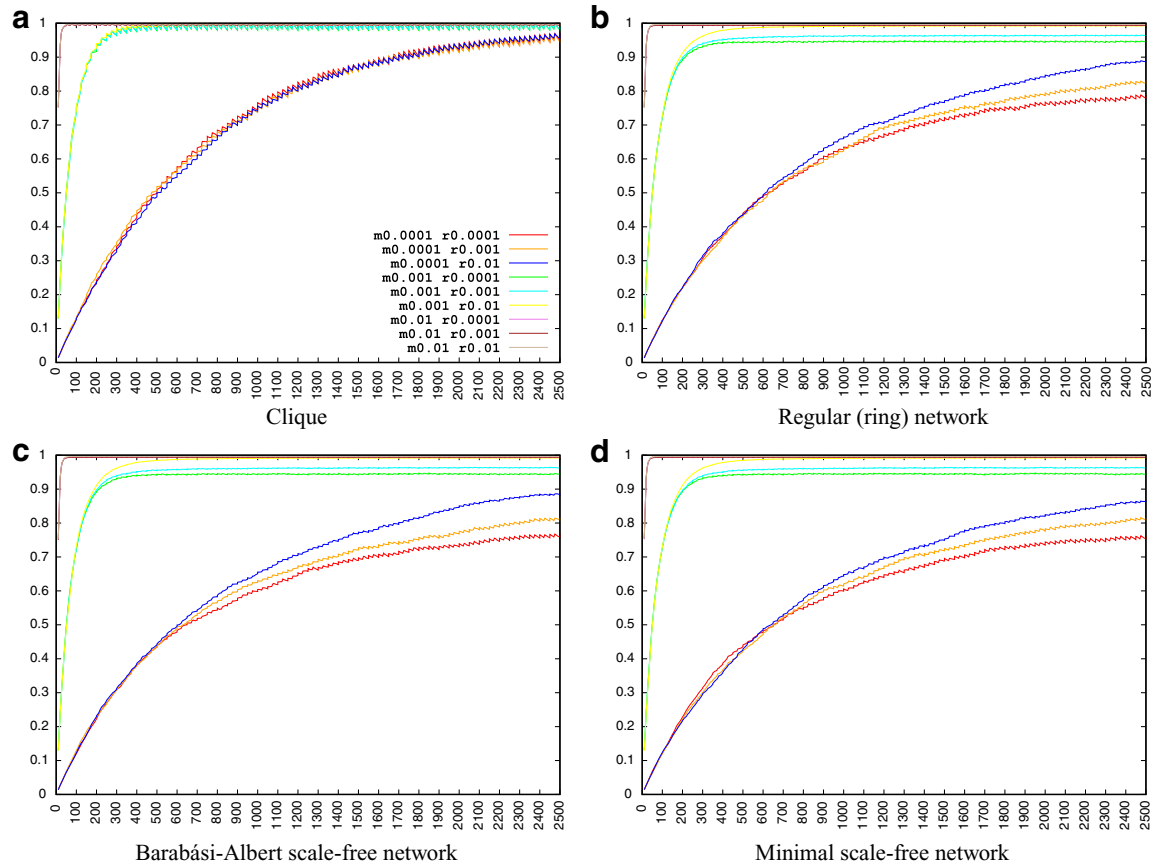


FIG. 2. Each subfigure (representing a different topology) contains a plot representing the average of the SID per generation, for each combination of Mutation and Recombination rates. See main text for details. SID, Simpson's Index of Diversity.

population size was 1000 per node, with each simulation containing 1,000,000 individuals. Over all, the mutation and recombination rate took values of 0.0001, 0.001, and 0.01. Figures 2 and 3 depict the results. In Figure 2, each image contains—for the four different topologies and nine different combinations of mutation and recombination rates—the average SID per generation. In Figure 3 each image contains—for the four different topologies and nine different combinations of mutation and recombination rates—the minimum, first quantile, second quantile (median), third quantile, and maximum SID per generation, presented as a box-and-whisker plot.

We first analyze the population diversity evaluation in a fully connected network (Figs. 2a and 3a), using it as a baseline, as it represents a well-mixed population. It is possible to observe that with higher rates of mutation, the rates of recombination and the effect of exchanges between nodes have no significant effect and the SID proliferates up to the point it stabilizes at the maximum value. For lower probabilities of mutation, there is a slight difference in what concerns the effect of the recombination rate and the effect of exchanges between nodes. When the recombination rate is also small, the SID increases slowly and, when it reaches the maximum value, small and recurrent fluctuations occur over time due to the exchange process.

As we increase the degree of heterogeneity, we begin to see some differences and the exchange process has a significant impact. The only constant observation for all topologies is that when the mutation rate is high, the diversity grows faster, reaching a stable state. When we adopt smaller values for mutation and recombination rates, the SID show pronounced fluctuations each time an exchange is executed. The most interesting case is when mutation and recombination rates are both 0.001, as we can see in Figures 2b and 3b, 2c and 3c, and 2d and 3d. In these cases, it is possible to observe that the interval of values that SID takes reflects that some nodes lose their diversity through time, even without any selection mechanism, as it is the case in this study. This can erroneously be attributed to selection events, suggesting that distinction between drift and selection is essential if we aim to understand natural evolutionary processes.

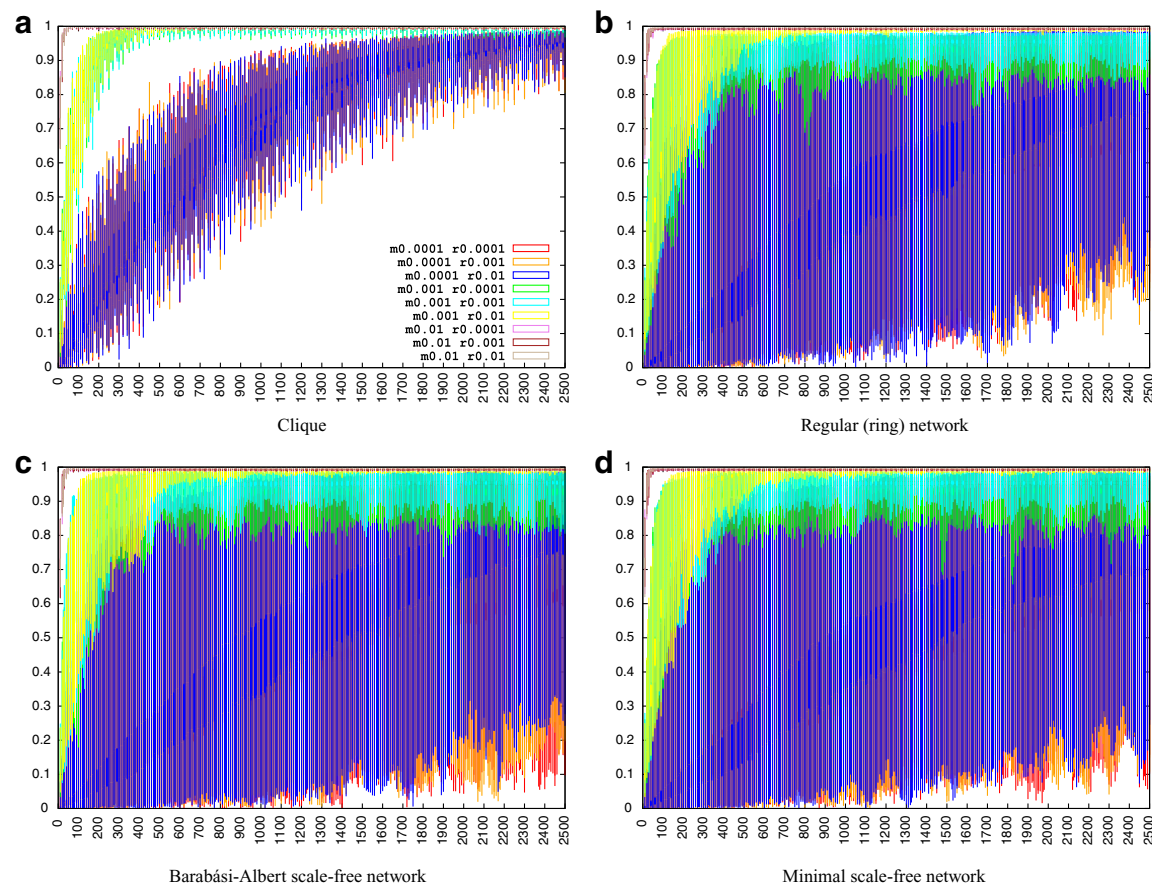


FIG. 3. Each subfigure (representing a different topology) contains a box-and-whisker plot (depicting the minimum, first quantile, second quantile (median), third quantile, and maximum values) of the SID per generation, for each combination of Mutation and Recombination rates. See main text for details.

5. CONCLUSIONS

We propose a framework for large-scale simulation of the evolution of bacterial population over complex networks, which can be run locally or in a distributed environment. We rely on the Wright–Fisher evolutionary model, and on the Map Reduce computational model on top of Apache Spark and GraphX API. The results show that we can run simulations with 1,000,000 individuals, for 250 generations in less than 3 minutes, in a distributed computing environment. Our results also point out that, for scale-free host contact networks—which represent most real-world scenarios—and according to Amdahl’s law, using more than 32 processing cores, does not lead to significant improvements for realistic simulations. Since the evolution of populations can occur independently, we may observe further gains from a large number of processing cores for larger populations, exploring the inherent parallelism. We leave this analysis for future work.

Platforms, like the one proposed in this study, have a wide range of applications. Large-scale studies and simulations are fundamental not only to understand bacterial population structure and the response of bacterial populations to selection pressures, but also to validate both models and methods, such as phylogenetic inference algorithms, and to understand how the sampling that is commonly done can affect or bias our perception of the true population structure. As an illustration of such applicability, we address the long-standing problem of genetic diversity due to neutral drift. According to our results, fluctuations in population diversity can be explained by neutral drift only, without selection pressure, strongly depending both on the average degree and degree of heterogeneity of the host contact network. As far as we know, such phenomena have not been observed before. As future work, we plan to use the proposed framework to evaluate the predictive capabilities of phylogenetic inference algorithms, which nowadays are being used to analyze increasingly large datasets, collected at multiple sites and pushing idealized evolutionary models to the limit.

ACKNOWLEDGMENTS

This work was partly supported by DEI, IST, Universidade de Lisboa, and national funds through FCT—Fundação para a Ciência e Tecnologia, under projects TUBITAK/0004/2014, LISBOA-01-0145-FEDER-016394 (SAICTPAC/0021/2015), PTDC/EEISII/5081/2014, PTDC/MAT/STA/3358/2014, SFRH/BD/129072/2017, and UID/CEC/50021/2013.

AUTHOR DISCLOSURE STATEMENT

The authors declare that no competing financial interests exist.

REFERENCES

- Amdahl, G.M. 1967. Validity of the single processor approach to achieving large scale computing capabilities. *In* Proceedings of the April 18–20, 1967, Spring Joint Computer Conference, AFIPS’67 (Spring), ACM, New York, NY. pp. 483–485.
- Barabási, A.-L., and Albert, R. 1999. Emergence of scaling in random networks. *Science* 286, 509–512.
- Dean, J., and Ghemawat, S. 2008. Mapreduce: Simplified data processing on large clusters, *Commun ACM* 51, 107–113.
- Dorogovtsev, S.N., Mendes, J.F., and Samukhin, A.N. 2001. Size-dependent degree distribution of a scale-free growing network. *Phys. Rev. E Stat. Nonlin. Soft. Matter. Phys.* 63(6 Pt 1), 062101.
- Fraser, C., Alm, E.J., Polz, M.F., Spratt, B.G., and Hanage, W.P. 2009. The bacterial species challenge: Making sense of genetic and ecological diversity. *Science* 323, 741–746.
- Fraser, C., Hanage, W.P., and Spratt, B.G. 2005. Neutral microepidemic evolution of bacterial pathogens. *Proc. Natl. Acad. Sci. U. S. A.* 102, 1968–1973.
- Fraser, C., Hanage, W.P., and Spratt, B.G. 2007. Recombination and the nature of bacterial speciation. *Science* 315, 476–480.
- Hanage, W.P., Spratt, B.G., Turner, K.M., et al. 2006. Modelling bacterial speciation. *Philos. Trans. R. Soc. Lond. B Biol. Sci.* 361, 2039–2044.
- Kimura, M. 1968. Evolutionary rate at the molecular level. *Nature* 217, 624–626.

- Lin, J., and Dyer, C. 2010. *Data-Intensive Text Processing with MapReduce*. Morgan and Claypool Publishers, San Rafael, CA.
- Maiden, M., Bygraves, J., Feil, E., et al. 1998. Multilocus sequence typing: A portable approach to the identification of clones within populations of pathogenic microorganisms. *Proc. Natl. Acad. Sci. U. S. A.* 95, 3140–3145.
- Ochman, H., Lawrence, J.G., and Groisman, E.A. 2000. Lateral gene transfer and the nature of bacterial innovation. *Nature* 405, 299–304.
- Page, L., Brin, S., Motwani, R., et al. 1999. The PageRank Citation Ranking: Bringing Order to the Web. Technical Report 1999–66, Stanford InfoLab, Stanford, CA.
- Robinson, D.A., Falush, D., and Feil, E.J. 2010. *Bacterial Population Genetics in Infectious Disease*. John Wiley & Sons, Hoboken, NJ.
- Simpson, E. 1949. Measurement of diversity. *Nature* 163, 688.
- Spratt, B.G., Hanage, W.P., and Feil, E.J. 2001. The relative contributions of recombination and point mutation to the diversification of bacterial clones. *Curr. Opin. Microbiol.* 4, 602–606.
- Teixeira, A.S., Monteiro, P.T., Carriço, J.A., et al. 2017. *Using Spark and GraphX to Parallelize Large-Scale Simulations of Bacterial Populations over Host Contact Networks*. Springer International Publishing, Cham, pp. 591–600.
- Tran, T.D., Hofrichter, J., and Jost, J. 2013. An introduction to the mathematical structure of the Wright-Fisher model of population genetics. *Theory Biosci.* 132, 73–82.
- Watts, D.J., and Strogatz, S.H. 1998. Collective dynamics of small-world networks. *Nature* 393, 440–442.
- Xin, R.S., Gonzalez, J.E., Franklin, M.J., et al. 2013. Graphx: A resilient distributed graph system on spark. In First International Workshop on Graph Data Management Experiences and Systems, GRADES'13, ACM, New York, NY. pp. 2:1–2:6.
- Zaharia, M., Chowdhury, M., Das, T., et al. 2012. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation, NSDI'12, USENIX Association, Berkeley, CA. p. 2.
- Zaharia, M., Chowdhury, M., Franklin, M.J., et al. 2010. Spark: Cluster computing with working sets. In Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing, HotCloud'10, USENIX Association, p. 10.
- Zhu, X., and Ghahramani, Z. 2002. Learning from labeled and unlabeled data with label propagation, Technical report cmu-cald-02-107, Carnegie Mellon University, Pittsburgh, PA.

Address correspondence to:

Dr. Andreia Sofia Teixeira
INESC-ID Lisboa and Instituto Superior Técnico
Universidade de Lisboa
Rua Alves Redol 9
Lisboa 1000-029
Portugal

E-mail: sofia.teixeira@tecnico.ulisboa.pt