*Bioinformatics*, 2021, 1–8
doi: 10.1093/bioinformatics/btab014
Advance Access Publication Date: 20 January 2021
Original Paper



Genome analysis

# Alignment-free Genomic Analysis via a Big Data Spark Platform

**Umberto Ferraro Petrillo [1,*], Francesco Palini[1], Giuseppe Cattaneo[2,†] and Raffaele Giancarlo[3,†]**

[1]Dipartimento di Scienze Statistiche, Università di Roma – La Sapienza, Rome 00185, Italy, [2]Dipartimento di Informatica, Università di Salerno, Fisciano (SA) 84084, Italy and [3]Dipartimento di Matematica ed Informatica, Università di Palermo, Palermo 90133, Italy

*To whom correspondence should be addressed.
†The authors wish it to be known that, in their opinion, the last two authors should be regarded as Joint Last Authors.
Associate Editor: Peter Robinson

Received on October 22, 2020; revised on December 28, 2020; editorial decision on December 31, 2020; accepted on January 6, 2021

## Abstract

**Motivation:** Alignment-free distance and similarity functions (AF functions, for short) are a well-established alternative to pairwise and multiple sequence alignments for many genomic, metagenomic and epigenomic tasks. Due to data-intensive applications, the computation of AF functions is a Big Data problem, with the recent literature indicating that the development of fast and scalable algorithms computing AF functions is a high-priority task. Somewhat surprisingly, despite the increasing popularity of Big Data technologies in computational biology, the development of a Big Data platform for those tasks has not been pursued, possibly due to its complexity.

**Results:** We fill this important gap by introducing FADE, the first extensible, efficient and scalable Spark platform for alignment-free genomic analysis. It supports natively eighteen of the best performing AF functions coming out of a recent hallmark benchmarking study. FADE development and potential impact comprises novel aspects of interest. Namely, (i) a considerable effort of distributed algorithms, the most tangible result being a much faster execution time of reference methods like MASH and FSWM; (ii) a software design that makes FADE user-friendly and easily extendable by Spark non-specialists; (iii) its ability to support data- and compute-intensive tasks. About this, we provide a novel and much needed analysis of how informative and robust AF functions are, in terms of the statistical significance of their output. Our findings naturally extend the ones of the highly regarded benchmarking study, since the functions that can really be used are reduced to a handful of the eighteen included in FADE.

**Availabilityand implementation:** The software and the datasets are available at https://github.com/fpalini/fade.

**Contact:** umberto.ferraro@uniroma1.it

**Supplementary information:** Supplementary data are available at *Bioinformatics* online.

## 1 Introduction

Alignment-free distance and similarity functions (AF functions, for short) have been introduced as an alternative to traditional alignment-based methods, e.g. (Altschul*et al.*, 1990; Smith and Waterman, 1981), in order to assess how similar each pair of sequences in a collection are to each other. By now, their use has been widely investigated for sequence analysis in genomics (Zielezinski*et al.*, 2019a), metagenomics (Benoit *et al.*, 2016) and epigenomics (Giancarlo *et al.*, 2015; 2018). The pros/cons of AF functions with respect to their alignment counterparts is well presented in Zielezinski*et al.* (2019a). One of the key pro features highlighted in that study is that their implementations offer data scalability, opportunities that alignment methods lack. Taking into account the throughput of HTS technologies, another compelling case on how important is to design and implement fast and scalable algorithms for the computation of AF functions is presented in Ondov*et al.* (2016). It is of interest here to notice that the mentioned two studies clearly indicate that the computation of AF functions is now a Big Data problem.

As such, it needs algorithmic solutions that use Big Data technologies to grant efficiency and scalability as a function of the available computational resources and of the amount of data to process. Somewhat surprisingly, although those technologies are finding more and more use in computational biology (Cattaneo*et al.*, 2019; Mushtaq and Al-Ars, 2015), and cloud storage and computing is the future of genomic data (Kahn, 2011), only a few studies in that area are available, concentrating on AF methods for informational and linguistic analysis of genomic sequences (Cattaneo*et al.*, 2017). Yet, an effective Big Data platform supporting both the computation of

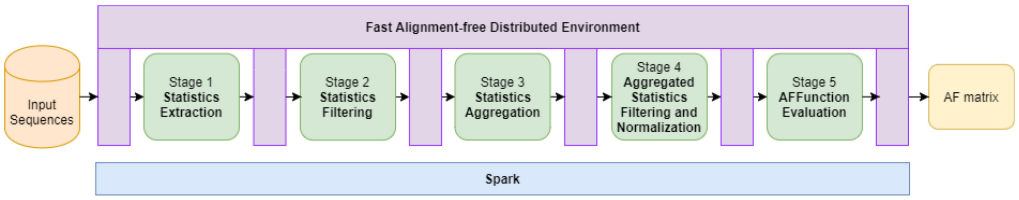


**Fig. 1.** Alayout of the logical architecture of the basic pipeline for the fast computation of AF functions

AF functions and based on a pervasive Big Data framework such as Spark, would place AF sequence comparison at a peer with other important domains in data science (see, e.g. Gonzalez *et al.*, 2012) that have such platforms, as well as contribute to increase their usage in the life sciences.

Our first contribution is to address this acute need. Indeed, we propose FADE, the first extensible and scalable Spark platform for the effective computation of AF functions. In its starting configuration, it offers eighteen implementations of highly performing AF functions according to results in Zielezinski*et al.* (2019a). The basic pipeline of our platform consists of five (possibly optional) stages, as outlined in Figure 1.Its key features are as follows.

- **FADE is user-friendly and easily extensible** FADE comes as a ready-to-use Spark application that can be easily executed without writing any line of code. If needed, its standard processing can be customized by just editing a provided reference configuration file, so as to choose from an included library which statistics to extract and which AF functions to evaluate. Some examples of configuration files are reported in Supplementary Section S3.1 of Supplementary Material. The user can add support for a target statistic or a target AF function not originally included in the library, by writing the corresponding code using one of the provided class templates. This part is outlined in Section 2.3.
- **FADE is scalable and efficient.** In order to assess FADE ability to profitably support the implementation of AF functions, we have compared two of the most prominent ones in the shared memory category, i.e. Mash (Ondov*et al.*, 2016) and FSWM (Leimeister*et al.*, 2017), versus their respective implementations supported by FADE (denoted with the prefix FADE). Being those latter based on a distributed framework, they are able to take advantage of the much higher number of processing cores available, achieving performances that are much better than those of shared memory tools and that scale as a function of the processing units available. It is to be noted that, given the wide range of application scenarios for AF sequence comparison, varying from the examination of a large collection of very small reads, up to the comparison of a few huge genomic sequences, a flexible workload assignment is fundamental to grant scalability and speed. To this end, FADE provides, and we have experimented with, three workload choices. Transparently from the user, each of the three is associated to a transformation of the logical basic pipeline into a suitable 'run time' software pipeline, which is then executed. The Partial Aggregation strategy results to be the most appropriate, with varying workloads. All the results, and the corresponding experiments, regarding this part are presented in Section 3.1.

Our second contribution, introduced in Section 2.5 and detailed in Section 3.2, shows the ability of FADE to tackle data and compute intensive tasks and it is of interest in its own right. Indeed, we use novel ideas, that boil down to very costly Monte Carlo Simulations, to gain insights into the properties of AF functions, going a step further in the direction indicated by the mentioned benchmarking study. The end result is new guidelines on the use of AF functions in day-to-day genomic analysis tasks.

To this end, we consider reliability and robustness of AF functions in terms of the statistical significance of the distance/similarity matrices they produce on benchmark datasets. In details, we consider a *P*-value obtained via a Monte Carlo simulation with the Null Hypothesis being that the values obtained by a given AF function on two biological sequences is no better than the value obtained on two random sequences. A value is significant when the Null Hypothesis is rejected. We account for repeated tests by applying Bonferroni Correction. Our key findings are the following.

- **A novel class of AF functions: consistently significant.** Across benchmark datasets from Zielezinski*et al.* (2019a), only a handful of the eighteen AF functions we have considered provide distance/similarity matrices for which the Null Hypothesis is rejected for the vast majority of their entries. We refer to those functions as *consistently significant*, since they behave consistently well, statistically, irrespective of the dataset, and it is well known that statistical relevance is a good indication of biological relevance (Dudoit and Fridlyand, 2002; Giancarlo *et al.*, 2008b; Leung *et al.*, 1996). They are members of the D2 family (Song et al., 2014) (D2 and D2*), and FSWM (Leimeister*et al.*, 2017). Since the D2 family has been studied extensively from the statistical point of view, our results confirm that it is an excellent choice. For FSWM, this analysis is completely new. This part is in Section 3.2.1.
- **Sensitivity to noise of consistently significant AF functions.** In order for the consistently significant AF functions to be reliably useful for 'everyday analysis', it is also important to assess how sensitive they are to the presence of 'noise'. That is, how the performance of an AF matrix varies as a function of the number of its entries that are not statistically significant. To the best of our knowledge, this sensitivity analysis is completely new and the findings are very informative. It is described in Section 3.2.2. Based on those results, a small amount of'noise' is enough to have a significant reduction in performance. Therefore, the operational range of those functions is limited to AF matrices that pass at least 95% of the statistical significance test. Interestingly, since we used phylogenetic tree construction for this sensitivity analysis, we find that UPGMA (Sneath and Sokal, 1973) is better than the Neighbor Joining method (Saitou and Nei, 1987) (NJ, for short), in dealing with small amounts of 'noisy' entries in the AF matrices. This aspect of those two methods is new.

In the reminder of this paper, we assume that the reader is familiar with Apache Spark (Apache Spark, 2016). For the convenience of the reader, and due to space limitations, a short primer is reported in Supplementary Section S1 of Supplementary Material.

## 2 Materials and methods

### 2.1 A selection of alignment-free distances and similarity functions

The two most popular classes of AF functions are those based on *k-mer statistics* and on *micro-alignments*. For this research, we consider those two classes, providing also motivation for our choices.

An AF function in the first class is based on *k-mer statistics* (or histogram statistics; Luczak*et al.*, 2019). and it can be used to analyze a set of sequences as follows. For each sequence in the set, the contiguous subwords of length $k$ therein contained (i.e.$k$-mers) with their associated frequencies are counted. The result is a set of vectors. Then, sequences are compared pairwise by computing suitable distance/similarity functions between each pair of vectors. The interested reader can find in Zielezinski*et al.* (2019a) a list of the ones that have been the object of a recent benchmarking study. One of the most surprising findings of that study is that those simple AF functions are among the best performing and most versatile in terms of application domain. We have chosen the best performing ones according to that benchmarking, representatives of all types of AF functions described in Luczak*et al.* (2019) and that can be broadly used in biological studies, e.g.metagenomics (Benoit *et al.*, 2016). The complete list of the selected AF functions is in Supplementary Section S2.1 of Supplementary Material, together with their definitions.

An AF function in the second class is based on the notion of *spaced word* match between two sequences. This latter is usually encoded via a binary vector, where the one entries indicate the positions where two subsequences of the two sequences must be identical. Zero entries may not matter. The distance between two sequences is estimated with the use of those matches. The interested reader can find an example of those methods in Denckert*et al.* (2020), together with relevant references. For our study, we have chosen the **FSWM** distance (Leimeister*et al.*, 2017), since it has emerged as the most competitive in this class of AF functions (Leimeister*et al.*, 2017; Zielezinski*et al.*, 2019a). Details regarding its definition are in Supplementary Section S2.2 of Supplementary Material.

## 2.2 A Spark platform for fast computation of AF functions: the basic pipeline

In this section, we first provide a 'user level' functional description of the proposed platform. Then we outline, again at a functional level, how the architectural issues influencing scalability have been dealt with.

## 2.3 A user-view of the basic pipeline as a general and extensible spark programming paradigm for implementation of AF functions

To a user, the basic pipeline appears as a succession of stages, described next. Assuming that the dataset to be processed is composed of $n$ sequences, the output is an $n \times n$ matrix, in which entry $(i, j)$ corresponds to the value of the chosen AF function on sequences $i$ and $j$. It is also worth pointing out that since the input sequences are partitioned over different computing nodes, two steps are required to collect a global statistic. First, the desired statistic is partially evaluated on each node holding a part of a given sequence. Second, all partial statistics are aggregated to derive the global statistic.

- *Stage 1: collection of partial statistics*. In this stage, the statistic that needs to be collected, e.g. $k$-mers, is extracted from each of the input sequences and provided as output.

This is transparently done in a distributed way, so that each computing node extracts the partial statistic from the parts of the input sequences it stores. We anticipate that Stage 3 takes care of aggregating the different partial statistics extracted from a same sequence. User code can be provided to support more statistics, in addition to those already included in the platform.

- *Stage 2: feature-based statistics filtering*. The user implementing the AF algorithm may require the exclusion of a selected subset of features from the statistics it is computing, e.g. specific $k$-mers such as those containing the 'N' character. To this end, this stage acts as a filter to exclude from the output of the previous stage the selected features, according to conditions specified by the user. The filtering occurs at this point, so as to (possibly) alleviate the workload of the following stages.
- *Stage 3: statistics aggregation*. All partial statistics extracted by different computing nodes during Stage 1 (and possibly Stage 2), but originating from the same input sequence are automatically and transparently gathered on a same node and aggregated. For instance, statistics about a particular $k$-mer and extracted from different parts of the same sequence are summed to obtain the overall $k$-mers statistics for that sequence. User code can be provided to support aggregation for statistics, in addition to those already included in the platform.
- *Stage 4: value-based aggregated statistics filtering and normalization*. Stage 2 filters the features existing in a statistic, while this stage filters according to a user-defined condition targeting the aggregated value assumed by a feature in a statistic. For instance, one would want to exclude low frequency $k$-mers when collecting $k$-mers statistics. This stage also performs, if required, data normalization. Indeed, as well argued in Luczak*et al.*, (2019), it is advisable to take the statistic of each sequence, e.g.$k$-mers counts, and transform it so that all the statistics refer to the same scale. Details are in Giancarlo *et al.* (2018) and Luczak*et al.* (2019).
- *Stage 5: AF matrix computation*. For each pair of different input sequences, their final aggregated statistics are sent by the platform to the same node. The AF function that has been chosen, from the ones available, is evaluated on each pair of sequences and the AF matrix is filled accordingly. In addition to the ones available, additional functions can be supported by providing user code.

The output AF matrix is encoded as a distributed data structure, whose content can be saved on file or used as input for further analysis.

Each of the aforementioned stages is modeled as one or more Spark distributed transformations. A general and extensible library of built-in basic functions implementing them is described in details in Supplementary Section S3.1 of Supplementary Material. The user interested in supporting a new statistic and/or implementing a variant of these functions can provide her code, as described in Supplementary Section S3.2 of Supplementary Material.

## 2.4 Architectural engineering: tuning the pipeline as a function of the workload

We briefly highlight the different data partitioning strategies supported by FADE, designed with the aim of tuning the basic pipeline as a function of the input workload so as to allow for an efficient and scalable execution. Additional details regarding them are available in Supplementary Section S4 of Supplementary Material, while their comparative experimental evaluation is reported in Section 3.1.

- *Strategy 1: total aggregation*. This strategy allows for very good execution times when extracting and processing statistics having an overall small size. This is possible because all statistics (either partial or aggregated) extracted during the pipeline are maintained and processed on a single node of the distributed system. The same occurs to the partial AF function evaluated on each statistic. On a one side, this implies that no distributed computation occurs, apart from that of Stage 1. On the other side, this strategy allows to avoid the data transmission overhead required to transfer data to the nodes of the distributed system prior to their processing.
- *Strategy 2: no aggregation*. This strategy allows for a very good scalability when extracting and processing statistics from very large input data. This is possible because every single statistic (either partial or aggregated) extracted during the pipeline is

managed as a stand-alone data object. The same occurs to the partial AF function evaluated on each statistic. The only aggregation occurs at the end of the pipeline when, for each pair of distinct sequences, partial AF function values are combined to return the overall value of the function. This ensures for a very fine scalability and load balancing as Spark tends to scatter these data objects uniformly at random on the different nodes of the distributed system. This holds because the amount of memory required to process single data objects is, typically, much smaller than the one required for processing collections of data objects.

- *Strategy 3: partial aggregation.* This strategy allows for a good trade-off between efficiency and scalability when extracting and processing statistics from large input data. This is possible because all statistics (either partial or aggregated) extracted during the pipeline are partitioned into bins. The same occurs to the partial AF function evaluated on each statistic. Consequently, each node processes a smaller number of data records batches (i.e. the content of each bin) rather than a (potentially) much larger number of single data records. This has a positive effect both on the processing and the communication times.

## 2.5 Reliability and robustness of AF functions

### 2.5.1 Reliability of an AF function via a hypothesis test Monte Carlo simulation

Intuition suggests that the larger the number of entries in the AF matrix not due to 'chance', the more indicative of biological relevance the outcome, e.g. phylogenetic tree, of that function use is expected to be. However, experience suggests that AF functions may have a 'behavior' that depends on the dataset being processed. Therefore, it is uncontroversially desirable to use functions that are consistently significant.

We formalize such an intuition by requiring that a consistent AF function must provide a high percentage of statistically significant entries in its corresponding AF matrix, with very little dependence on the input dataset. In regard to those tests, we resort to a Monte Carlo Simulation method, in which we assume as the Null Hypothesis $H_0$ that two biological sequences are as similar as two random ones. An entry $(i, j)$ of an AF matrix is statistically significant when the Null Hypothesis involving the two sequences associated to that entry is rejected with a given significance level. The entire simulation consists of three steps, described next. The Spark algorithms implementing it are briefly described in Supplementary Section S5 of Supplementary Material. In what follows, we consider only the case of similarities, since the case of distances is analogous.

*Step 1: synthetic datasets generation via bootstrapping.* The first step is to devise a procedure that generates synthetic datasets, that are meant to represent 'random data'. Such a task can be accomplished by choosing a *Null Model*, e.g. an information source emitting symbols uniformly and at random. However, in our case, it seems more appropriate to resort to *bootstrapping* (Efron, 1979), i.e. to generate the synthetic datasets from real ones, since it is desirable to preserve the biological origin of the input dataset also in the synthetic ones. To this end, we proceed as follows.

Let $S$ be the input dataset and let $q$ be a parameter. All $q$-mers of sequences in $S$ are extracted and placed in a bin $B$. Then, in order to obtain a synthetic dataset $\hat{S}$, we extract uniformly and at random $q$-mers from $B$ in order to form new sequences to be included in $\hat{S}$. This latter has the same number of sequences as in $S$ and each sequence in $\hat{S}$ corresponds to only one in $S$ in terms of length.

It is to be noted that the parameter $q$ allows us to generate synthetic datasets along a wide spectrum of subsequence statistics present in $S$, e.g. $q = 1$ corresponds to the case in which the synthetic dataset is generated according to the empirical probability distribution of symbols in $S$.

*Step 2: significance test via a Monte Carlo simulation for two sequences.* The next step consists of the following simulation, adapted from Giancarlo *et al.* (2008a). It applies to sets of two sequences. $C$ denotes the AF function to which the procedure is applied. The Null Hypothesis $H_0$ is that two input sequences are as similar as two random ones, when similarity is assessed via $S$. We want to reject the Null Hypothesis with confidence level $\alpha$, with the performance of $\ell$ Monte Carlo Simulations.

**Procedure MECCA**$(\ell, C, S, \alpha)$

1. For $1 \leq i \leq \ell$, compute a new set of two sequences $\hat{S}_i$ according to the procedure outlined in Step 1. Compute the similarity between the two sequences in $\hat{S}_i$ via $C$. Let $T_i$ be its value.
2. For $1 \leq i \leq \ell$, sort the $T_i$ values in non-decreasing order and let $SL$ be the corresponding list.
3. Let $T$ denote the value of $C$ computed on $S$. Let $j$ be the maximal index such that $SL[j] < T$. Let $\delta = (j/\ell)$. The $P$-value is then $1 - \delta$ and, and letting $\alpha$ be the desired significance level, the hypothesis that the two sequences in $S$ are as similar as two randomly chosen ones is rejected with that significance level if $1 - \delta \leq \alpha$.

*Step 3: AF matrix significance via Bonferroni correction.* Consider now a set $S$ consisting of $n$ sequences, labeled from 1 to $n$. Let $F$ be the $n \times n$ AF matrix for $S$ computed via $C$. In order to assess how statistically significant is $F$, with family-wise significance level $\alpha$, we can resort to the pairwise application of the simulation procedure outlined in Step 2. Since we are performing $m = \frac{n(n-1)}{2}$ hypothesis tests, we have to correct for rejecting $H_0$ simply by chance. Since those tests may not be assumed to be independent, we use the well-known Bonferroni correction. That is, for each test performed for each entry of $F$, $H_0$ is rejected with significance level $\alpha/m$. Then, we reject the Null Hypothesis that the matrix is no better than one obtained on a set of random sequences, if all $m$ entries pass the test. However, even if the full matrix does not pass the test, such a procedure outlines entries that are statistically significant in terms of similarity values of the corresponding sequences.

### 2.5.2 Robustness of an AF function via a matrix perturbation method

In addition to the reliability of an AF function, it is also important to assess how robust is the function with respect to 'noise', i.e. the number of entries in the AF matrix that are not statistically significant. Informally, we refer to this as the *operational range* of an AF function. In order to empirically estimate this latter, we informally proceed as follows: the larger is the amount of noise injected in a AF matrix returned by a given function, the worse should be the biological relevance of its outcome (e.g. the phylogenetic tree). Then, to measure the operational range of a function, we start from the AF matrix returned by that function on a given dataset and assume as a reference its performance score. Finally, we study its performance variation, assuming it will decrease while increasing the amount of noisy entries. More precisely, we assume that the dataset under consideration has a gold standard solution. For this study, it is a phylogenetic tree associated to the species in the dataset. We also use a computational method $G$ to build a phylogenetic tree from an AF matrix and a distance/similarity measure $D$ to assess how different is a tree produced by $G$ via an AF matrix with respect to the gold standard. Formal details of the basic perturbation step follow.

- Given the AF matrix computed by $C$ on dataset $S$, we select uniformly and at random a given percentage of entries and substitute each value with a 'noisy' one. For histogram-based functions, such a value is taken randomly among the ones appearing in the AF matrices generated via Monte Carlo simulation. As for FSWM, since the AF matrices it returns when evaluated on the synthetic datasets are likely to contain null values (see discussion about filtering in Section 3.2.1), the 'noisy' value is obtained by increasing the original one by a value chosen uniformly and at random. For both types of AF functions, the new matrix so obtained is used to build a tree via $G$, and its distance from the gold standard is computed via $D$. The loss in

performance at the given noise level is given by the difference in the $D$ score obtained with the noisy versus the original matrix.

## 3 Results and discussion

As a first preliminary step, we selected for our study datasets coming from Zielezinski*et al.* (2019a), with the criterion that the AF functions chosen here would work on them [see again (Zielezinski*et al.*, 2019a)]. They are reported in Supplementary Section S6 of Supplementary Material. As a second preliminary step, we assessed that our AF functions implementations are in line with those used for the benchmarking of the AFproject (Zielezinskiet al., 2019b) (details are omitted for brevity and available upon request). All our experiments have been executed on the hardware platform described in Supplementary Section S6 of Supplementary Material. The parameters of the algorithms have been set according to the procedures described in Supplementary Section S7 of Supplementary Material. Execution times have been measured by collecting the job execution elapsed time returned within the Spark framework.

### 3.1 Assessing the scalability of methods supported by FADE and the effectiveness of the aggregation strategies

#### 3.1.1 Experiments

When considering a single computing platform where multiple computing units coexist on the same motherboard and communicate, at no cost, by using shared memory, nothing can beat the performance of a native application purposely developed to take advantage of that setting. On the other hand, the number of processing units and memory size on a single motherboard is a natural performance bottleneck.

Given the above scenario, the use of a distributed framework is to overcome such a bottleneck, by scaling performance as a function of the number of computing units far over the ones on a stand-alone computing platform. Therefore, it is important to assess the efficiency of methods supported by our software pipeline, when compared to analogous state-of-art shared memory alignment-free tools, while assessing the scalability of our framework. For such an analysis, we have chosen two reference software systems that provide shared memory parallel software for their evaluation: FSWM (Leimeisteret *al.*, 2014) and Mash (Ondovet *al.*, 2016).

We measure the execution time of FSWM and Mash, as well as that of their implementations supported by our framework, i.e. FADE-FSWM and FADE-Mash, on the Plants (assembled) dataset and while increasing the level of parallelism, as explained shortly. Mash and FSWM have been executed on a single machine equipped with 8 computing cores. The FADE versions have been executed on a distributed framework equipped with 24 worker nodes, for a total of 128 computing cores. Each of those machines is identical to the one on which the shared memory algorithms have been executed. The result of this experiment is visible in Figure 2, where we report the execution time of these applications as a function of the number of concurrent threads, for shared memory applications, or of Spark workers, for FADE. Each thread/worker executes on a single core.

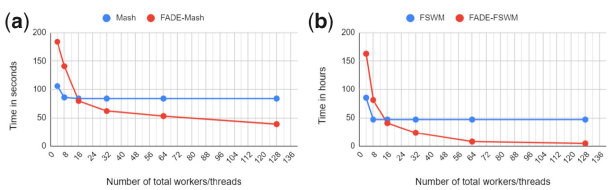


**Fig. 2.** Execution time, on the ordinate axis, required by both versions of (**a**) Mash and (**b**) FSWM on the Plants (assembled) dataset, using an increasing number of workers/threads. The Partial Aggregation strategy has been used, since naturally suited to the algorithmic methods supporting Mash and FSWM, respectively

In order to assess the effectiveness of the aggregation strategies supported by FADE, we executed a significance test for all of the histogram-based functions, with different data sizes. The results are reported in Table 1.

#### 3.1.2 Results

The results of the first experiment show that the shared memory versions of Mash and FSWM are faster than the implementations supported by FADE, when using a very small number of threads compared to workers. This is expected. With the increase of the number of those units, the performance gap narrows. As soon as the use of 16 threads/workers is reached, FADE-Mash and FADE-FSWM require about the same execution time as that of their shared memory counterparts. However, those latter stop scaling, due to the bottleneck mentioned earlier, while the FADE methods continue to scale as more threads/workers are added.

The results of the second experiment clearly show that the Partial Aggregation strategy is the most flexible of the three, since it guarantees equal or better performance with respect to the other two, independently of the workload.

### 3.2 Reliability and robustness of AF functions: an application of FADE to data/compute intensive analysis

#### 3.2.1 Reliability of AF functions

*The experiment*. For each of the benchmark datasets included in this study, we execute the AF statistical significance test described in Section 2.5.1. Specifically, for each matrix obtained via a dataset, we generate additional ones via bootstrapping. Then, we reject the null hypothesis family-wise with $P$-value $\leq 1\%$, applying Bonferroni correction to all of its $m$ entries. The number $l$ of simulations for each test has been chosen according to the size of the dataset being processed, so as to guarantee the execution of the experiments in a reasonable amount of time. An outline of these parameters, including the choice of $k$ and the values of $q$, is available in Supplementary Section S7 of Supplementary Material.

A summary of the results is shown in Supplementary Figure S10 of Supplementary Material. For each dataset, each AF function and each considered null model, the percentage of entries passing the test is reported. This value is drawn in green, if at least 75% of the entries passes the test, in red, if no entry passes the test, and in yellow, in the remaining case. Figure 3 reports a representative synopsis of those results: a dataset where most AF functions perform well, one in which many perform poorly and one in which most perform poorly.

*Results and insights*

- *A novel class of AF functions: consistently significant*. With reference to Figure 3 and Supplementary Figure S10 of Supplementary Material, it is evident that there are AF functions returning matrices passing the family-wise significance test either fully or with a high percentage of entries, for all of the benchmark datasets. We denote them as *consistently significant*, since they behave consistently well, independently of the dataset and,

**Table 1.** Execution time, in minutes, required by our framework to execute one instance of the AF significance test, for all of the histogram-based functions, on three reference datasets with different aggregation strategies

| Dataset | No aggregation | Partial aggregation | Total aggregation |
|---|---|---|---|
| Small | 0.9 | 0.9 | 1.1 |
| Large | 2.73 | 2.82 | NA |
| Very large | 48.3 | 42.0 | NA |

*Note*:The NA value indicate that the test took too long to complete and was stopped. Small = Mithocondria (assembled). Large = Shigella (assembled). Very large = Plants (assembled).

Mithocondria (assembled)

| | q=1 | q=7 | q=10 |
|---|---|---|---|
| Canberra | 100. | 100. | 100. |
| Chebyshev | 3.3 | 3. | 3.7 |
| ChiSquare | 100. | 100. | 99.3 |
| D2 | 100. | 100. | 99. |
| D2S | 100. | 99.7 | 81.7 |
| D2Z | 100. | 99.7 | 98.7 |
| D2* | 100. | 91.3 | 82. |
| Euclidean | 89.3 | 85.7 | 84.3 |
| FSWM | 100. | 100. | 100. |
| Harmonic Mean | 98.7 | 70.7 | 48. |
| Intersection | 65. | 40. | 35.3 |
| Jaccard | 27.3 | 28. | 43.3 |
| Jeffreys | 99.3 | 96. | 95. |
| Jensen Shannon | 99.7 | 96.3 | 95. |
| Kulczynski2 | 100. | 99.3 | 99. |
| Manhattan | 100. | 100. | 99.3 |
| Mash | 78. | 86. | 83.7 |
| Squared Chord | 100. | 100. | 99.3 |

E.coli (unassembled, coverage=1)

| | q=1 | q=7 | q=10 |
|---|---|---|---|
| Canberra | 100. | 100. | 100. |
| Chebyshev | 0. | 0. | 0. |
| ChiSquare | 58.4 | 21.9 | 16.3 |
| D2 | 100. | 100. | 100. |
| D2S | 100. | 100. | 100. |
| D2Z | 100. | 79.8 | 65. |
| D2* | 100. | 100. | 100. |
| Euclidean | 1.5 | 1.5 | 1.5 |
| FSWM | 100. | 100. | 100. |
| Harmonic Mean | 100. | 96.3 | 85. |
| Intersection | 100. | 15.5 | 2.5 |
| Jaccard | 0. | 0. | 0. |
| Jeffreys | 71.9 | 28.1 | 15.3 |
| Jensen Shannon | 82.5 | 34.7 | 17.5 |
| Kulczynski2 | 13.3 | 3.7 | 3.7 |
| Manhattan | 66.5 | 28.1 | 18.2 |
| Mash | 0. | 0. | 0.9 |
| Squared Chord | 41.1 | 20.7 | 16.3 |

Plants (assembled)

| | q=1 | q=7 | q=10 |
|---|---|---|---|
| Canberra | 22. | 22. | 27.5 |
| Chebyshev | 0. | 0. | 0. |
| ChiSquare | 6.6 | 6.6 | 6.6 |
| D2 | 98.9 | 97.8 | 97.8 |
| D2S | 14.3 | 7.7 | 7.7 |
| D2Z | 1.1 | 1.1 | 1.1 |
| D2* | 100. | 100. | 100. |
| Euclidean | 0. | 0. | 0. |
| FSWM | 100. | 100. | 100. |
| Harmonic Mean | 19.8 | 18.7 | 18.7 |
| Intersection | 42.9 | 29.7 | 27.5 |
| Jaccard | 1.1 | 3.3 | 3.3 |
| Jeffreys | 2.2 | 2.2 | 2.2 |
| Jensen Shannon | 3.3 | 2.2 | 2.2 |
| Kulczynski2 | 2.2 | 2.2 | 2.2 |
| Manhattan | 13.2 | 11. | 11. |
| Mash | 2.2 | 2.2 | 2.2 |
| Squared Chord | 6.6 | 6.6 | 6.6 |

**Fig. 3.** Summary of the Hypothesis Test results for the different AF functions considered in this research when executed on three different datasets with $q = 1, 7, 10$ and with significance level set to 1%. Datasets have been chosen as described in the Main Text

**Table 2.** Corruption tables relative to D2, D2* and FSWM functions and q = 1, varying dataset, method for building the phylogenetic tree, metric to evaluate the distance between the reference and obtained tree

| | | D2 | | | | | D2* | | | | | FSWM | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Mitochondria | Shigella | Yersinia | | | Mitochondria | Shigella | Yersinia | | | Mitochondria | Shigella | Yersinia |
| RF | NJ | 6 (200%) | 10 (90.9%) | 4 (Inf%) | RF | NJ | 5 (125%) | 7 (63.6%) | – | RF | NJ | 17 (425%) | 20 (500%) | – |
| | UPGMA | – | – | – | | UPGMA | – | – | – | | UPGMA | 10 (333.3%) | 6 (120%) | – |
| MCM | NJ | 28 (23.3%) | 23 (26.4%) | 14 (Inf%) | MCM | NJ | 15 (14.6%) | 7 (8.0%) | – | MCM | NJ | 40 (40.8%) | 69 (97.2%) | 4 (33.3%) |
| | UPGMA | −31 (-31.0%) | −39 (-24.2%) | – | | UPGMA | – | 5 (6.3%) | – | | UPGMA | 49 (245%) | 41 (146.4%) | – |

*Note*: In each entry, it is reported the difference between the phylogenetic distance evaluated on the distance matrix with 10% of corrupted entries and the original distance matrix (between the brackets is reported the percentage difference, indicating with 'Inf' when the tree obtained from the original distance matrix is equal to the reference tree, i.e. the distance between the trees is 0). Dashed entries represent cases where the introduction of noise caused only a small change in distance, as specified in the Main Text.

as already stated in the Introduction, it is well known that statistical relevance is a good indication of biological relevance (Dudoit and Fridlyand, 2002; Giancarlo *et al.*, 2008b; Leung *et al.*, 1996). They are: D2, D2*, FSWM. The remaining ones either perform inconsistently or poorly (Chebyshev).

- *Filtering can be useful.* While the statistical guarantees of AF functions in the D2 family are well known and have been identified via deep investigations (Song *et al.*, 2014), we find a novel fact regarding FSWM: it is quite good in delivering matrices that pass the statistical significance test. This can be attributed to the filtering mechanism present in the algorithm and that was designed to 'flush out' the parts of the statistics it is collecting and that are considered 'weak'. Such a filtering is able to detect the 'low relatedness' of the synthetic genomes ensuring that the original genomes 'win' the test.

### 3.2.2 Robustness of AF functions: the case of consistently significant functions

*The experiment.* We concentrate only on AF functions that have been classified as consistently significant in the previous section, since they are the most likely to be useful on a day-to-day basis. For each of them, this experiment is conducted by injecting an increasing percentage of noisy entries into an AF matrix, following the method outlined in Section 2.5.2. As a computational method $G$ to build a phylogenetic tree from an AF matrix, we use both UPGMA and NJ. This choice is motivated by the fact that, with pros and cons, those two methods are reference points in the literature. Moreover, as a distance/similarity measure $D$ to assess how different is a tree produced by $G$ via an AF matrix with respect to the gold standard, we use both the Matching Cluster (Bogdanowicz and Giaro, 2013) (MCM for short) and the Robinson Fould (Robinson and Foulds, 1981) (RF, for short) metrics. Such a choice is motivated by the fact that they were among the top performing ones in a classic benchmarking study (Kuhner and Yamato, 2015), in particular the second. As for RF, it is quite sensitive to small changes in tree topology, i.e. it 'saturates' rapidly, a fact that has been criticized in the past (Penny *et al.*, 1982). Yet, it is a standard in the literature.

The results of this experiment are available in Supplementary Figures S11–S16 of Supplementary Material, where we plot the 'distance' from the gold standard as the number of noisy entries grows. Moreover, to gain more insights, we also report in Tables 2–4, the variation in distance only when 10% noisy entries are injected. Moreover, we do not consider changes in the distance from the gold standard that are in the interval $\pm 3$, i.e. reasonably close to the case of 'no noise'.

*Results and insights*

- *AF matrices are sensitive to the injection of noisy entries.*Supplementary Figures S11–S16 of Supplementary Material show a trend of deterioration in classification ability as the percentage of noisy entries increases. Such a trend is more pronounced when we use the RF metric to assess how close to the gold standard is a phylogenetic tree computed from a noisy matrix with respect to the one that uses an uncorrupted AF matrix. This is possibly due to the 'saturation' effect, but MCM largely confirms those experiments.
- The results in Tables 2–4 give more compelling evidence of the sensitivity to noise of the AF functions under analysis. In particular, FSWM is quite sensitive, with both NJ and UPGMA. As for the other two AF functions, they display such a sensitivity mostly

**Table 3.** Corruption tables relative to q = 7

| | | D2 | | | | | D2* | | | | | FSWM | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Mitochondria | Shigella | Yersinia | | | Mitochondria | Shigella | Yersinia | | | Mitochondria | Shigella | Yersinia |
| RF | NJ | 4 (133.3%) | 11 (100%) | 4 (Inf%) | RF | NJ | 6 (150%) | 7 (63.6%) | – | RF | NJ | 17 (425%) | 20 (500%) | – |
| | UPGMA | – | – | – | | UPGMA | – | – | – | | UPGMA | 10 (333.3%) | 4 (80%) | – |
| MCM | NJ | 22 (18.3%) | 31 (35.6%) | 14 (Inf%) | MCM | NJ | 21 (20.4%) | 67 (77.0%) | – | MCM | NJ | 40 (40.8%) | 100 (140.8%) | 4 (33.3%) |
| | UPGMA | –32 (–32.0%) | –28 (–17.4%) | – | | UPGMA | – | 5 (6.3%) | – | | UPGMA | 49 (245%) | 32 (114.3%) | – |

*Note*: Refer Note in Table 2.

**Table 4.** Corruption tables relative to q = 10

| | | D2 | | | | | D2* | | | | | FSWM | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Mitochondria | Shigella | Yersinia | | | Mitochondria | Shigella | Yersinia | | | Mitochondria | Shigella | Yersinia |
| RF | NJ | 4 (133.3%) | 10 (90.9%) | 4 (Inf%) | RF | NJ | 7 (175%) | 7 (63.6%) | – | RF | NJ | 17 (425%) | 19 (475%) | – |
| | UPGMA | – | – | – | | UPGMA | – | – | – | | UPGMA | 10 (333.3%) | 4 (80%) | – |
| MCM | NJ | 22 (18.3%) | 24 (27.6%) | 14 (Inf%) | MCM | NJ | 22 (21.4%) | 19 (21.8%) | – | MCM | NJ | 40 (40.8%) | 87 (122.5%) | 4 (33.3%) |
| | UPGMA | –32 (–32%) | –24 (–14.9%) | – | | UPGMA | – | 5 (6.3%) | – | | UPGMA | 49 (245%) | 33 (117.9%) | – |

*Note*: Refer Note in Table 2.

when used with NJ. Or, better, UPGMA seems to be able to tolerate a small amount of noise in the input data. We believe that such an aspect of UPGMA is novel. It is to be noted that the entries with negative values, i.e. an improvement in classification with the addition of random entries, is justified by the fact that the original classification was so poor that even noise could do no harm.

- In conclusion, the indication we receive from the above is to use D2, D2* and FSWM in conjunction with UPGMA and with a high number of statistically significant entries in the AF matrix.

## 4 Conclusion

We have provided the first Spark platform, supporting AF functions evaluation, that is extensible and that guarantees good scalability with respect to the workload and computational resources available. This is witnessed by the very good performance, both in terms of computing time and scalability, of FADE-Mash and FADE-FSWM when compared to their state-of-the-art shared memory counterparts. This is particularly relevant, since the shared memory version of Mash has a leadership position due to its speed.

In order to demonstrate the ability of FADE to support large data and compute intesive tasks, we have conducted a novel analysis of AF functions. Thanks to it, we gain insights into which functions are best suited for day-to-day AF genomic analysis: D2, D2* and FSWM. We also account for the 'operational range' of those latter measures, i.e. circumstances in which a result due to those methods can be trusted. Such an 'operational range' translates into the requirement to use the AF matrices corresponding to those methods when the vast majority of their entries is statistically significant. Moreover, the use of UPGMA is recommended, since it is less sensitive to 'noise' in the input data, compared to NJ.

## Acknowledgements

R.G. was grateful to Prof. Chiara Romualdi for helpful discussions. All authors thank the Department of Statistical Sciences of University of Rome—La Sapienza for computing time on the TeraStat cluster and for other computing resources, and the Error! Hyperlink reference not valid. for having made available a cutting edge OpenStack Virtual Datacenter for this research.

## Funding

G.C., R.G. and U.F.P. were partially supported by GNCS Project 2019'Innovative methods for the solution of medical and biological big data'. R.G. was additionally supported by MIUR-PRIN project 'Multicriteria Data Structures and Algorithms: from compressed to learned indexes, and beyond' [2017WR7SHH]. U.F.P. and F.P. were partially supported by Università di Roma—La Sapienza Research Project 2018'Analisi, sviluppo e sperimentazione di algoritmipraticamenteefficienti'.

*Conflict of Interest*: none declared.

## References

Altschul,S.F. *et al.* (1990) Basic local alignment search tool. *J. Mol. Biol.*, **215**, 403–410.

Apache Software Foundation. (2016) Apache Spark. http://spark.apache.org (1 Novermber 2020, date last accessed).

Benoit,G. *et al.* (2016) Multiple comparative metagenomics using multiset k-mer counting. *PeerJComput. Sci.*, **2**, e94.

Bogdanowicz,D. and Giaro,K. (2013) On a matching distance between rooted phylogenetic trees. *Int. J. Appl. Math. Comput. Sci.*, **23**, 669–684.

Cattaneo,G. *et al.* (2017) An effective extension of the applicability of alignment-free biological sequence comparison algorithms with Hadoop. *J. Supercomput.*, **73**, 1394–1417.

Cattaneo,G. *et al.* (2019) MapReduce in computational biology via Hadoop and Spark. In: Ranganathan,N.S. *et al.* (eds.) *Encyclopedia of Bioinformatics and Computational Biology*, **Vol. 1**. Elsevier, Oxford, pp. 221–229.

Dencker,T. *et al.* (2020) Multi-SpaM: a maximum-likelihood approach to phylogeny reconstruction using multiple spaced-word matches and quartet trees. *NAR Genomics Bioinf.*, **2**, lqz013.

Dudoit,S. and Fridlyand,J. (2002) A prediction-based resampling method for estimating the number of clusters in a dataset. *Genome Biol.*, **3**, research0036.1.

Efron,B. (1979) Bootstrap methods another look at the jackknife. *Ann. Stat.*, **7**, 1–26.

Giancarlo,R. *et al.* (2008a) A tutorial on computational cluster analysis with applications to pattern discovery in microarray data. *Math.Comput. Sci.*, **1**, 655–672.

Giancarlo,R. *et al.* (2008b) Computational cluster validation for microarray data analysis: experimental assessment of clest, consensus clustering, figure of merit, gap statistics and model explorer. *BMC Bioinformatics*, **9**, 462.

Giancarlo,R. *et al.* (2015) Epigenomic k-mer dictionaries: shedding light on how sequence composition influences nucleosome positioning *in vivo*. *Bioinformatics*, **31**, 2939–2946.

Giancarlo,R. *et al.* (2018) In vitro versus in vivo compositional landscapes of histone sequence preferences in eucaryotic genomes. *Bioinformatics*, **34**, 3454–3460.

Gonzalez,J.E. *et al.* (2012) Powergraph: distributed graph-parallel computation on natural graphs. In: *Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, Hollywood, CA, USA, USENIX Association, pp. 17–30.

Kahn,S.D. (2011) On the future of genomic data. *Science*, **331**, 728–729.

Kuhner,M.K. and Yamato,J. (2015) Practical performance of tree comparison metrics. *Syst. Biol.*, **64**, 205–214.

Leimeister,C.-A. *et al.* (2014) Fast alignment-free sequence comparison using spaced-word frequencies. *Bioinformatics*, **30**, 1991–1999.

Leimeister,C.-A. *et al.* (2017) Fast and accurate phylogeny reconstruction using filtered spaced-word matches. *Bioinformatics*, **33**, 971–979.

Leung,M.Y. *et al.* (1996) Over- and underrepresentation of short DNA words in herpesvirus genomes. *J. Comput. Biol.*, **3**, 345–360.

Luczak,B.B. *et al.* (2019) A survey and evaluations of histogram-based statistics in alignment-free sequence comparison. *Brief. Bioinf.*,**20**, 1222–1237, 12.

Mushtaq,H. and Al-Ars,Z. (2015) Cluster-based Apache Spark implementation of the GATK DNA analysis pipeline. In:*2015 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. Washington, DC, USA, IEEE, pp. 1471–1477.

Ondov,B.D. *et al.* (2016) Mash: fast genome and metagenome distance estimation using minhash. *Genome Biol.*, **17**, 132.

Penny,D. *et al.* (1982) Testing the theory of evolution by comparing phylogenetic trees constructed from five different protein sequences. *Nature*, **297**, 197–200.

Robinson,D.F. and Foulds,L.R. (1981) Comparison of phylogenetic trees. *Math.Biosci.*,**53**, 131–147.

Saitou,N. and Nei,M. (1987) The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Mol. Biology Evol.*,**4**, 406–425.

Smith,T.F. and Waterman,M.S. (1981) Identification of common molecular subsequences. *J. Mol. Biol.*, **147**, 195–197.

Sneath,P.H.A. and Sokal,R.R. (1973) *Numerical Taxonomy. The Principles and Practice of Numerical Classification*.W.H. Freeman and Company, San Francisco, USA.

Song,K. *et al.* (2014) New developments of alignment-free sequence comparison: measures, statistics and next-generation sequencing. *Brief. Bioinf.*,**15**, 343–353.

Zielezinski,A. *et al.* (2019a) Benchmarking of alignment-free sequence comparison methods. *Genome Biol.*, **20**, 144.

Zielezinski,A. *et al.* (2019b) Afproject. http://afproject.org (1 Novermber 2020, date last accessed).