MANUAL.md

FADE

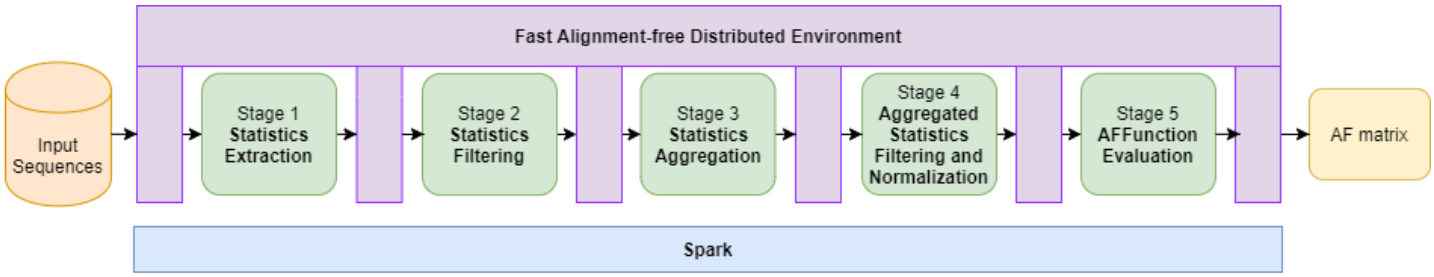
FADE (Fast Alignment-free Distributed Environment) is an extensible framework, developed to efficiently compute alignment-free (AF) functions on a set of large genomic sequences. It runs over *Apache Spark* (>=2.3, https://spark.apache.org/) and requires a *Java* compliant virtual machine (>= 1.8, https://adoptopenjdk.net). Moreover, it can perform Monte Carlo hypothesis test simulations, in order to analyze the AF matrix computed on the set of sequences.

A copy of FADE, including source code, can be downloaded from: FADE

The processing pipeline

AF analysis conducted with FADE use a distributed processing pipeline of five stages. For each stage, the user has the possibility to define which module to use among the ones available with the FADE standard library, as well as coding a new module. The stages are:

1. *StatisticsExtractor*: The statistics that needs to be collected, e.g. k-mers, are 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 statistics from the parts of the input sequences it stores.
2. *StatisticsFilter*: Filters from the list of partial statistics the ones satisfying a provided boolean condition.
3. *StatisticsAggregator*: All partial statistics extracted by different computing nodes during Stage 1 but originating from the same input sequence are automatically and transparently gathered on a same node and aggregated.
4. *AggregatedStatisticsFilter*: Filters from the list of aggregated statistics the ones satisfying a provided boolean condition.
5. *AFFunctionEvaluator*: 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.



Usage

The software package includes a single executable jar file, **fade-1.0.0-all.jar**, that can be used to run tests from the command line. It can be run by using the standard *Apache Spark* `spark-submit` command with the following syntax:

```
spark-submit fade-1.0.0-all.jar [conf-file]
```

If `conf-file` is not specified, the program will look for a `fade.conf` file in the working directory. FADE has two main operational tasks:

- *distance evaluation*: a set of AF matrices is evaluated starting from a collection of input genomic sequences and according to a provided list of AF functions;
- *montecarlo simulations*: run a statistical significance test on a set of input AF functions with respect to a given collection of genomic sequences

Quickstart

For a quick start, assuming both Apache Spark and Java are properly installed, move in the package root directory and run the following command:

```
spark-submit fade-1.0.0-all.jar quickstart.conf
```

As a result, FADE will be used to evaluate the k-mer based Euclidean AF function, with k=5, on the Mitocondria dataset (included in this package). The resulting distance matrix will be saved in the *output* directory. The settings of this experiment are defined in the *quickstart.conf* configuration file.

Configuration File

The creation of a proper configuration file is the recommended way to instruct *FADE* about the experiments to perform, as it does not require any programming skill. Alternatively, it is possible to write a Java application using he FADE facilities in thanks to the provided programming API (see the `Main.java` class for an example).

Basic Parameters

In the following table we report a list of the main parameters that can be set in a FADE configuration file.

Parameter	Description
`task`	The task that *FADE* has to perform. It can be `distance` (distance/similarity evaluation, default) or `simulation` (montecarlo simulations).
`local`	It has to be `true` (default) if *Apache Spark* is run in local mode. `false`, if run on a cluster.
`input`	The path of the file containing the sequences. If more files are considered, the directory path can be specified.
`output`	The path of the output files.
`assembled`	The value can be `long` (default), if each input file contains only one long assembled sequence; `short`, if each input file contains one or more short assembled sequences; `no`, if each input file contains one or more reads of an unassembled genome.
`type`	The value can be `fasta` (default) if the sequences given as input are in *FASTA* format, or `fastq` if they are in *FASTQ* format.
`extractor`	The name of the statistics *Extractor* to use (see below).
`aggregator`	The name of the statistics *Aggregator* to use (see below).
`evaluator`	The name of the AF function *Evaluator* to use (see below). Several comma-separated names can be provided.
`strategy`	The aggregation strategy to apply. The value can be `no_aggregation`, `partial_aggregation` (default) or `total_aggregation`.
`k`	The length, in terms of nucleotides, of the statistics to collect.
`slices`	The number of parallel tasks. It should match at least the number of Spark workers. The default value is *128*.
`split_size`	The size of the chunks used when partitioning the input sequences into smaller parts (default is *100,000* nucleotides. If '0', no partitioning occurs
`buffer_size`	Used when reading short sequences. It defines the maximum size (in bytes) allowed for these sequences (default is *4096*).
`filter_regex`	The regular expression used to filter a subset of statistics (Stage 2 of *FADE*).
`aggr_filter_regex`	The regular expression used to filter a subset of the aggregated statistics (Stage 4 of *FADE*).

Example 1

In the following example, the k-mer based D2 and Euclidean AF functions are evaluated (with k=5) on the *mitochondria* dataset of long genomic sequences stored in the data/mito directory. The outcoming AF matrices are saved in the *dist_mito*. In this case the *no aggregation* strategy is used, with a support for a maximum number of 192 parallel tasks.

```
task=distance

k=5

local=false
slices=192
strategy=no_aggregation

input=data/mito
output=dist_mito

extractor=fade.kmer.KmerExtractorByStatistic
aggregator=fade.kmer.KmerAggregatorByStatistic
evaluator=fade.affunction.D2, fade.affunction.Euclidean
```

Example 2

In the following example, the k-mer based Canberra and Jaccard AF functions are evaluated (with k=12) on the *ecoli* dataset of reads, with coverage 0.0125, stored in the data/ecoli_0125 directory. A *Monte Carlo* simulation with 5 samples is performed. In this case the *partial aggregation* strategy is used, with a support for a maximum number of 32 parallel tasks. Here it is used a local *Spark* execution. The output is an evaluation of the AF matrices computed.

See Monte Carlo Simulations and MC Simulations output for details.

```
task=simulation

k=12
simulations=5
q=10

assembled=no
local=true
slices=32
strategy=partial_aggregation

input=data/ecoli_0125
output=sim_ecoli_0125_q10_n10

extractor=fade.kmer.KmerExtractorByBin
aggregator=fade.kmer.KmerAggregatorByBin
evaluator=fade.affunction.Canberra, fade.affunction.Jaccard
```

Kmer-based Statistics configuration parameters

FADE includes a reengineered implementation of the FastKmer package for the efficient collection of kmer-based statistics. Its configuration parameters are:

Parameter	Description
`x`	The (k,x)-mers compression factor, used by *Fastkmer*.
`m`	The signature length, used by *Fastkmer*.
`canonical`	It is `true` (default) if canonical kmers are considered, otherwise `false`.

An example of configuration file using *FastKmer* follows:

```
task=distance

x=3
m=4

local=true
k=7
assembled=long
slices=64
strategy=partial_aggregation

input=data/shigella
output=dist_shigella

extractor=fade.kmer.fast.FastKmerExtractorByBin
aggregator=fade.kmer.fast.FastKmerAggregatorByBin
evaluator=fade.distance.D2Distance, fade.distance.EuclideanDistance
```

Kmer-based Statistics with MinHash approximation configuration parameters

FADE includes an implementation of the Mash algorithm, based on the extraction of representative sketches for the fast distance estimation, by using the MinHash technique. Its configuration parameters are:

Parameter	Description
`s`	The sketch size, used by *Mash*.
`canonical`	It is `true` (default) if canonical kmers are considered, otherwise `false`.

An example of configuration file using *Mash* follows:

```
task=distance

k=5
s=1000

local=false

input=data/mito
output=dist_mito

extractor=fade.mash.MashExtractor
aggregator=fade.mash.MashAggregator
evaluator=fade.affunction.Mash
```

Spaced word-based Statistics configuration parameters

FADE includes a distributed implementation of the FSWM algorithm, that is based on *spaced words* statistics. When using this algorithm, the following additional parameters are available:

Parameter	Description
`pattern`	The binary pattern used by *FSWM* to distinguish among care and don't care positions.
`threshold`	A threshold used by *FSWM* to filter only relevant matches.

An example of configuration file using *FSWM* follows:

```
task=distance
```

```
k=20

pattern=10001011010100101001
threshold=0

local=false
slices=192
strategy=partial_aggregation

input=data/mito
output=dist_mito

extractor=fade.sw.SwExtractorByBin
aggregator=fade.sw.SwAggregatorByBin
evaluator=fade.affunction.FSWM
```

Monte Carlo Simulations

For more information about the AF statistical significance test based on Monte Carlo simulations, see the paper Alignment-free Genomic Analysis via a Big Data Spark Platform.

The available parameters that can be set when running this type of task are:

Parameter	Description
`q`	The parameter used for the synthetic datasets generation.
`simulations`	The number of simulations to perform.
`significance`	If the value is `true` (default), it is evalued the significance of the matrix entries. See MC Simulations Output for details.

An example of configuration file where a significance test is run using 10 simulations follows

```
task=simulation

simulations=10
q=1

x=2
m=3

k=5

local=false
slices=192
strategy=partial_aggregation

input=data/mito
output=simul_mito_n10_q1

extractor=fade.kmer.fast.FastKmerExtractorByBin
aggregator=fade.kmer.fast.FastKmerAggregatorByBin
evaluator=fade.affunction.D2, fade.affunction.Euclidean
```

Output Files

AF Functions Output

The AF matrices returned by executing a set of AF functions on a collection of input genomic files is encoded as a a *csv* file where the rows represent genome pairs and each column represents one of the considered AF functions. An entry (i,j) of the output contains the value resulting from the application of the AF function j on the i-th genomes pair.

MC Simulations Output

The output of the MC-based significance test of a set of AF functions on a collection of input genomic files is encoded as a a *csv* file where the rows represent genome pairs and each column represents one of the considered AF functions. The content of each entry depends on the value of the configuration parameter `significance`:

- `significance=true`: each entry of the AF matrix is evaluated, considering the *Bonferroni correction*, returning **1** if the entry passes the hypothesis test. Otherwise it is **0**.
- `significance=false`: each entry contains the rank of the original genomes pair in respect to the synthetic ones, considering their AF function values.

When running a MC-based significance test, a *simulation_bkp* directory is automatically created and used to store temporary files. If the simulation is interrupted for whatever reason, it will be possible to resume by running it again and thanks to the files stored in this directory.

Supported Statistics and AF functions

Actually, two types of statistics, described as follows, are natively supported by the framework.

K-mer Statistics

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 AF functions, between each pair of vectors.

Available K-mer based AF functions

For each AF function, it is reported the full name of the Java class to be used for recalling that function in *FADE*.

- Canberra Distance (`fade.affunction.Canberra`)
- Chebyshev Distance (`fade.affunction.Chebyshev`)
- χ^2 Distance (`fade.affunction.ChiSquare`)
- D_2 Similarity (`fade.affunction.D2`)
- D_2^S Similarity (`fade.affunction.D2S`)
- D_2^Z Similarity (`fade.affunction.D2Z`)
- D_2^* Similarity (`fade.affunction.D2star`)
- Euclidean Distance (`fade.affunction.Euclidean`)
- Harmonic Mean Similarity (`fade.affunction.HarmonicMean`)
- Intersection Similarity (`fade.affunction.Intersection`)
- Jaccard Similarity (`fade.affunction.Jaccard`)
- Jeffrey's Divergence (`fade.affunction.Jeffrey`)
- Jensen-Shannon Divergence (`fade.affunction.JensenShannon`)
- Kulczynski2 Similarity (`fade.affunction.Kulczynski2`)
- Manhattan Distance (`fade.affunction.Manhattan`)
- Mash Distance (`fade.affunction.Mash`)
- Squared-Chord Distance (`fade.affunction.SquaredChord`)

Word Matches Statistics

Given a binary pattern *P* of match (1) and don't care (0) characters, there exists a *spaced word matching* between two sequences *s*, *t*, respectively in positions *i1* and *i2*, according to the pattern *P* with length *l*, if for each match position *m* in *P*, it is true that s[i1 + m] = t[i2 + m].

Available Word Matches based AF functions

- FSWM Distance (`fade.affunction.FSWM`)

Datasets

FADE has been extensively tested by using the following genomic datasets:

- E.coli/Shigella (39MB)
- Mitocondria (0.2MB)
- Plants (1.3GB)
- Unassembled E.coli (coverage 0.03125, 0.125, 1) (309MB)
- Unassembled Plants (coverage 1) (2.7GB)
- Yersinia (11MB)

Running on Amazon Web Services (AWS)

FADE has been designed to take fully advantage of the *HDFS* distributed file system, even thanks to the adoption of the *FASTdoop* library. For this reason, it requires input data to be stored on *HDFS*.

We refer the interested reader to the following link for a quick guide about the installation of *Apache Spark* on a free *EC2 AWS* instance: https://dzone.com/articles/apache-spark-setting-up-a-cluster-on-aws.

Build FADE from the source code

FADE has been developed as *Gradle* project (https://gradle.org/). Assuming you are inside the main project directory, in order to build the source code and create an executable jar you need to launch the following commands:

```
chmod +x gradlew  # needed only for Linux and MacOS
./gradlew shadowJar
```

Alternatively, you can import the project as a *Gradle* project into your IDE. In order to create the executable jar you need to click on the `shadowJar` task, listed among the *Gradle* tasks.

The task creates the **fade-1.0.0-all.jar** inside the *build/libs* directory.

Extensions

Supporting more Statistics

The user can add support for a target statistic not originally included in the library by extending and specializing the following classes:

- `Statistic` : provides a Java representation for the target statistic.
- `Value` : provides a Java representation for the partial or aggregated value assumed by the target statistic.
- `Data` : provides a Java representation for a collection of occurrences of the target statistic.

In order to implement a *no aggregation strategy* it is necessary to extend the following Java classes:

- `StatisticExtractorByStatistic`

- `StatisticAggregatorByStatistic`

In order to implement a *partial aggregation strategy* it is necessary to extend the following Java classes:

- `StatisticExtractorByBin`
- `StatisticAggregatorByBin`

In order to implement a *total aggregation strategy* it is necessary to extend the following Java classes:

- `StatisticExtractorByUnique`
- `StatisticAggregatorByUnique`

See *src/main/java/fade/kmer* directory as an example.

Supporting more AF Functions

In order to implement a new AF function, it is necessary to extend the `AFFunctionEvaluator` Java class and implement the following methods:

- `evaluatePartialAFValue`
- `combinePartialAFValues`
- `finalizeAFValue`

If the AF function to implement is a similarity, it is necessary to implement the `Similarity` interface, in order to rank correctly the genome pairs during the *Monte Carlo Simulations*.

See *src/main/java/fade/affunction* directory as an example.

Accessing Configuration Parameters

When a new *extractor*, *aggregator* or *evaluator* is implemented, can be useful to access the configuration parameters. It can be achieved requesting for the `Configuration` object using the `getConf()` method. The methods of the `Configuration` class are the following:

- `getString(name[, default])`
- `getInt(name[, default])`
- `getLong(name[, default])`
- `getBoolean(name[, default])`
- `getDouble(name[, default])`

In order to create a new configuration parameter, it can be simply set inside the configuration file. It is parsed and saved by *FADE* into the `Configuration` object.

The following is a list of parameters set by *FADE* during the computation and accessible with the `Configuration` object.

Parameter	Description
`n`	The number of sequences.
`len_s<id>`	The length of sequence *id*.
`countA_s<id>`	The number of nucleotide *A* in sequence *id*.
`countC_s<id>`	The length of nucleotide *C* in sequence *id*.
`countG_s<id>`	The length of nucleotide *G* in sequence *id*.
`countT_s<id>`	The length of nucleotide *T* in sequence *id*.
`distinct`	The number of distinct statistics. Available only after the aggregation stage.

Parameter	Description
`pseudo_s<id>`	The number of pseudocounts in sequence *id*. Available only after the aggregation stage and when `CountValue` is used.
`stdev_s<id>`	The standard deviation of sequence *id*. Available only after the aggregation stage and when `CountValue` is used.