Data Stream Analysis in Sliding Windows: Random Sampling and other Problems

January 19, 2022

Tutor: Conrado Martínez
Author: Francesc Ruiz Tuyà

Contents

1 Introduction

In many data stream applications we need to perform some analysis in a "window" or subsequence of contiguous elements, quite often the last M elements seen or the elements seen in the last X time units. For example, we might be interested in obtaining a random sample of the distinct elements seen in the last 10 minutes, or estimate how many distinct elements have been processed among the last 100000 processed items. Given the restrictions in processing time and memory available, exact solutions become unfeasible and we seek for randomized algorithms which are fast, have low memory requirements and provide probabilistic guarantees. In this project we will implement some of the algorithms available in the literature and conduct extensive experiments to assess their performance and compare their relative merits; we will also develop novel and original algorithms or variants of existing algorithm to compare them with the state-of-the-art solutions. We will mostly focus in algorithms to obtain random samples, a fundamental task for more complex statistical inference: detecting outliers, finding frequent items, detecting unusual patterns, etc.

2 Description of the project

This project is about improving an Affirmative Sampling Algorithm. Affirmative Sampling belongs to a category of algorithms called random samplers or probabilistic structures that acquire a subset of items from a data stream preserving the properties of the original data stream thus allowing estimations over the original dataset. This types of algorithms can accelerate the processing of data streams by orders of magnitude allowing for real time processing of big data. This algorithms are extremely efficient in terms of memory and computation time. The objective of this project will be to implement and test an improvement over the original algorithm. This improvement will consist in implementing a so called *moving window* to remove old elements from the subset in order to keep it up to date. Furthermore, this improvement will make some of the original algorithm properties to no longer be preserved thus some research will be needed in order to determine if the properties are completely lost or if they can still be estimated.

2.1 Areas of application

This project can be applied to the following areas [11]:

1. Network traffic analysis: DoS/DDoS attacks, worms, . . .
2. Database query optimization
3. Information retrieval: computing similarity index
4. Data mining
5. Recommendation systems

2.2 Practical example

Big companies such as Amazon are interested in website unique number of visitors over a certain period of time. If we assumed that each unique user has a unique IP address, we would need to calculate the number of unique IP addresses which are in a 128-bit string domain according to the IPv6 protocol. On March 2017, the top three most popular websites in the USA (Amazon, Ebay and Walmart) had 1.44 billion visits which take 192 GB of memory to store. Even the subset of unique users would require about 23 GB of memory meaning that performing any search over such a big set would be extremely inefficient [7].

2.3 Potential users

This algorithm could be beneficial to most users of technology. Even though most of us have never heard of this types of algorithms, we are already using them. Social networks are the main representative of this technology but other examples would be antiviruses, search engines, cookies, trackers, internet traffic, etc. But in general, every human interaction that generates huge volumes of noncritical data that just requires a good enough answer is the perfect target for this technology. We are not willing to wait 10 seconds for each Google search or database request. Companies don't want to invest 10 times as much for an increased performance of less than 2%. People don't want to wait 10 hours scanning all the files on a computer just to know whether there is a virus. Basically, all users that don't require an exact answer to their problems will benefit from this technology. Specially in this day an age where deeplearning is starting to pick up much attention, data gathering and data processing will also be essential to the development of deeplearning technologies

3 Setup

In order to set up the environment to execute experiments it's really easy, from the root directory, execute: *makefile load_environment* from the root directory This way of setting up the environment was based on [13]. For accessing the menu just type *python main.py* and soon the menu will appear.

4 Terms and concepts

4.1 Goal of the algorithm

Given a data stream of distinct elements we want to obtain a subset of elements that has the same statistical composition as the original datastream in order to measure a set of properties about it. This properties could be the number of distinct elements, the number of k-elephants, the number of k-mice, etc. Independently from the property we want to extract from the original dataset, we know that probabilistic .Random sampling algorithms are a compromise between accuracy and efficiency. They provide estimations of the real values of the properties to be measured but they do so in a fraction of the time and space that an exact solution would require. This algorithms also provide boundaries for maximum error expected from the estimation which means they provide mathematical guarantees. [11]

4.2 Mathematical concepts

- **Sequence of elements:**
  $\mathcal{Z} = z_1, z_2, ..., z_N$
- **Domain of the elements:**
  $\{\mathcal{U} | z_i \in \mathcal{U}\}$

- **Cardinality of the sequence (number of elements):**
  $\mathcal{N} = |\mathcal{Z}|$
- **Frequency of the sequence:**
  $f(\mathcal{Z}) = \mathfrak{n} \Big| Z = \{x_1 \circ f_1, x_2 \circ f_2, ..., x_\mathfrak{n} \circ f_\mathfrak{n}\}$
- **Algorithm's available memory (number of Bytes):**
  $\mathcal{M}$

4.3 Algorithm constraints

1. The algorithm can only process each element of the stream once. This means that the data stream acts like a queue and not like a vector.
2. Each element should be processed fast. This means that processing one element should have a cost of $\Theta(1)$.
3. The size of the memory used by the algorithm is an order of magnitude below the cardinality of the data stream;
   expressed by: $O(\mathcal{M}) << O(\mathcal{N})$.
   An ideal algorithm in terms of memory would use $M = \Theta(1)$ Bytes,
   a good algorithm $M = \Theta(\log \mathcal{N})$ Bytes.
4. No statistical assumption is made about the characteristics of the data stream or the sub-parts of it. That's where the hashing functions become an useful tool to avoid prejudice.
5. The unbiased estimator must be accurate and have a small standard error.

[11]

4.4 The evolution of random sampling algorithms

The algorithm was developed by G. Nigel Martin and mathematically documented by Philippe Flajolet. This collaboration resulted in the publication in 1984 of the article "Probabilistic Counting Algorithms for Data Base Applications" [6]. Later it has been refined in "LogLog counting of large cardinalities" by Marianne Durand and Philippe Flajolet, and "HyperLogLog: The analysis of a near-optimal cardinality estimation algorithm" by Philippe Flajolet et al.

In their 2010 article "An optimal algorithm for the distinct elements problem", Daniel M. Kane, Jelani Nelson and David P. Woodruff give an improved algorithm, which uses nearly optimal space and has optimal $O(1)$ update and reporting times.

4.5 Original algorithm: Probabilistic counting

Uses a hash function to transform each distinct item into a value $\in [0, 1]$[16]. This value is used as a way to choose a random subset of items from the data stream. Since each item has the same odds of being picked, the subset will be an accurate representation of the real sample thus maintaining many properties and proportions. Then a range of hash values and size of the subset is established meaning that if an item comes into the subset, another one has to go and the range of hashes required to enter the subset is reduced. [6] [11]

5 Tasks

5.1 Defining the project layout

This task consists in setting up a modular and scalable platform that takes into account all future requirements of the project. I will be developing the whole project in python since it's the most commonly used language when it comes to big data and deep learning which are the two areas that this algorithm could really helpful in.

5.1.1 Modular algorithms

The platform must allow the implementation and integration of new algorithms.

5.1.2 Modular hash functions

The platform must allow the use of specific hash functions independently of the algorithm.

5.1.3 Modular properties

The platform must allow the implementation and introduction of new properties that measure characteristics of both the datastream and the subset.

5.1.4 Modular data stream generator

The platform must allow the implementation of different ways to generate an input for the algorithms.

5.1.5 Verbose input parser

The execution of the script has to use natural language to make easier the comprehension of the type of test we are running.

5.1.6 Verbose output

Defining a good output format for our tests that makes the interpretation of results easy.

5.2 Improving the algorithm

This task must implement a dynamic subset that can increase or decrease in size according to a criterion that has yet to be established.

5.2.1 Optimizing data structures

The structures used in this new algorithm must be thoroughly planned an mathematically analyzed in order to be as optimal as possible within the constrictions of the selected programming language.

5.2.2 Implementing the dynamic subsets in the sliding window algorithm

Using the previous structures, develop a new algorithm that uses a moving time window. This algorithm will have unknown properties that haven't been mathematically demonstrated yet. This means that a lot of testing will be required to understand those properties and extract some conclusions. This task will be the main focus of this project since the rest of the tasks will depend on it.

5.3 Testing the algorithm

Once the algorithm works, it will be time to test if the results follow the mathematical rules. This will help me to find the bugs in the code and to compare the performance of distinct implementations. Both the algorithms and the testing method have to be really efficient since we are dealing with big data which will inherently take a lot of time to deal with. This task will also be really important since it will let us see any patterns in the data that could lead to mathematical conclusions. In order to achieve those conclusions, we will compare the improved sliding window algorithm with an already proven algorithm such as affirmative sampling.

5.3.1 Implementing previous algorithms

Once the improved algorithm works, it will be time to test it against other proven algorithms to see the differences.

5.3.2 Implement the required properties

The mathematical properties implemented in the layout phase won't be complete and some will have to be added.

5.3.3 Implement the required generators

In the previous tasks I will have already developed a basic generator but when reaching the testing phase maybe there is a need to improve the generation methodology of data streams depending on the observations.

5.3.4 Optimize the testing structures

In order to reduce testing time, the testing algorithm will have to be thoroughly planned. This includes the generator, all the properties, maybe some time measurements and the gathering of data. It must be stressed that the goal of this project is not the creation of an optimal algorithm but rather the creation of a proof-of-concept testing platform. This is why python is the selected language and not C or C++ since it lets the programmer develop new ideas much much faster. Despite all this, from a theoretical viewpoint, all structures will be optimal or quasi-optimal within the limits of the programming language.

5.3.5 Extracting conclusions

From the results obtained from the testing phase I will be able to understand the strengths and weaknesses of the newly created algorithm. I will then document all this conclusions using statistical and mathematical proof.

5.4 Setting a goal for the project

Initially I couldn't set a goal for the project since I didn't know whether the windowed algorithm would have similar properties to the original one. From the tests I did with the first prototype in the middle of November I could graphically observe a trend in the collected data. It appeared as if the two algorithms were not the same but they had a high similarity in the intersection of subsets. Since we are dealing with a lot of randomness, it's really hard to prove this similarity through mathematical means. That's why I chose the empirical approach. If I'm able to explore a big enough space and the novel algorithm is consistently similar to the original algorithm then this hypothesis can be accepted. This is an important thesis since it would improve the performance of running a windowed algorithm by orders of magnitude depending on the size of the window. If the window was of 1,000,000 of elements it would mean that the windowed algorithm would process the stream 1,000,000 times faster than the Affirmative Sampling algorithm and it would have the same mathematical guarantees.

- Null hypothesis: the Affirmative Sampling algorithm executed at $[t - window\_size, t]$ and the Affirmative Sampling Windowed algorithm are equivalent. This means that all the properties of the Affirmative Sampling algorithm which have already been mathematically proven can be extended to this new Affirmative Sampling Windowed algorithm.
- Collect data: this task has proven to be the most difficult one so far since I had to define a way to explore as much of the domain as possible in a repeatable, modular and efficient way (both in terms of memory and CPU usage).
- Perform a statistical test: the current version of the project lacks this part although it will be the next one to be implemented. Since I already have all the data collected it will only be a matter of processing it and extracting conclusion.
- Reject or fail the hypothesis

5.5 Documenting

I will choose an incremental documentation method meaning that I will be implementing and documenting my work simultaneously. Despite this, documenting will have it's own task.

5.6 Meetings with the project tutor

Weekly meetings with my tutor will be set up in order to discuss the progress and assess the situation and course of action.

6 Architecture

This project has been a big challenge in terms of architectural design. Because the goal of the project is to prove a conjecture, the program had to answer the following requirements:

- Fast: we are working with big data, performance bottlenecks are specially dangerous in this environment.
- Robustness: in order for the code to be robust we need to be able to detect errors easily.
- Scalable: if the code is too convoluted it's much harder to implement new features and the features you implement tend to have more bugs.
- Repeatable: the conclusions obtained should be easily validate by repeating the experiment and reaching the same conclusions.

6.1 Instance Serialization

In order for the code to be robust I implemented full experiment serialization. As it can be observed in the Annex 20.3, when executing an experiment, before even instantiating the objects, on import time, the classes can automatically and recursively generate a json file which contains all the fields that are user modifiable. This json could have been created manually and modified with each change in the architecture but this would have made refactoring much harder and would have caused scalability issues. Furthermore, by having this json and building all objects using it, we can easily save and load user configurations making the results easily repeatable by providing execution certificates. The Serialization of the objects uses 4 classes on the Utils/ folder and is based on the python Decorator principle as explained in Chapter 7 of [3]. Object serialization has been one of the hardest tasks on this project but the end result is really good.

6.2 Builders

Algorithms, Hashs and Generators all have Builder classes. This was done in order to prevent cyclic imports when serializing all the objects. Builder classes are used to implement multiple inheritance problems. On the serialized jsons the attribute named '_builder_catalog' serializes all the children that can be built by a certain builder.

7 Implementation of the Controller

At first I developed a prototype application which was already quite modular but it had some parts that were just a mess. That's why I decided to clean the code and create a second application where the structure was much cleaner and I also followed the Model View Controller (MVC) software design pattern as shown in figure 1. The whole project also tries to follow PEP 8 guidelines [2] to generate a code easily understood and pythonic. I also follow generic guidelines such as inheritance in order to try to reuse the code as much as possible by implementing common functions at the highest scope possible. This means that this project has an architecture that could be scaled really easily.

7.1 Algorithms

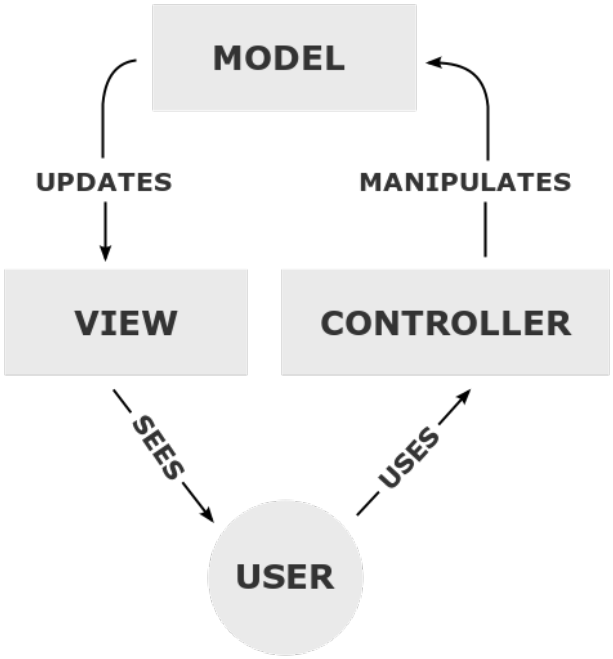
7.1.1 Affirmative Sampling

The basic algorithm developed by Conrado Martínez (my tutor) [11]. This algorithm has some great properties mathematically demonstrated such as the ability to estimate the cardinality of the real data stream. The cost of processing a new item from the stream is $O(1)$

7.1.2 Affirmative Sampling Windowed

A version of the Affirmative Sampling algorithm that in addition of acting like the normal Affirmative Sampling algorithm also deletes old elements. This behaviour means that this algorithm should have a much more updated picture of the data stream since we won't take old elements into account. The cost of processing a new item from the stream is $O(1)$

Figure 1: MVC software design pattern



7.1.3 Affirmative Sampling Buffered

Affirmative Sampling Buffered: an algorithm that tries to imitate the "Affirmative Sampling Windowed algorithm but is just the basic Affirmative Sampling algorithm with some minor tweaks used to prove the null hypothesis. The cost of processing a new item from the stream is $O(window\_size)$. The trade off of being really inefficient is that this algorithm has the same strong mathematical background as the Affirmative Sampling algorithm.

7.2 Generator

It's a class that implements an iterable object with an infinite ammount of elements (evaluated in a lazy way). I have implemented two kinds of generator.

7.2.1 ZipfGenerator

This generator is used to simulate human speech. As shown in [5], normal adult human speech can be characterized by a Zipf random generator with a β of 2.0. This article also explains how schizophrenics have different betas. If the schizophrenic person has an incoherent discourse, talking about many different topics, it usually repeats words much less than a normal human thus having a $\beta \in [2.11, 2.42]$. On the other hand obsessive schizophrenics or children with obsessive disorders or limited vocabulary tend to have a discourse with a lower β than normal: $1 < \beta < 2$. Because the Zipf law can be observed in many situations of nature I thought it was a good type of generator to test the characteristics of the algorithms before trying more realistic datasets that might be more random and offer less stability. Currently the implementation of this generator uses a vocabulary of 500,000 distinct words to imitate the English vocabulary but the 1000 most common ones are not taken into account since they offer no meaning due to the high frequency of use. The output of this generator consists of integers where the value represents the rank.

7.2.2 TextGenerator

This generator parses a text file and loops over it as many times as necessary outputting one word each time.

7.3 Evaluator

This class receives a Generator and two Algorithms and a status. It then starts requesting words from the generator and feeding them to the algorithms while saving the progress in a single file. The status is the set of parameters that can affect the result which can be also called domain or epochs.

7.4 Epoch

This class encapsulates a set of inner for's that iterate over a set of iterator objects. For example if we have an iterator with 10 elements, another one with 2 elements and a third one with 7 elements, the number of different statuses that would be outputted by this class would be $10 \times 2 \times 7 = 140$. Each of this status will be one of the input parameters of the Evaluator meaning that in this example, 140 files would be generated. All the files have a naming convention based on which index each dimension is currently at when executing the Evaluator.

7.4.1 EpochIter

This class encapsulates all the possible hardcoded iterators which are *range* and *linspace*. This allows the user to modify numerical values of the configuration.

7.5 Experiment

This class iterates over the epochs and initializes the different evaluators.

7.6 Timer

Contains a wrapper function used to log execution times of each module. That information is currently ignored but it doesn't affect the execution of the code since the amount of information stored is constant.

7.7 Hash

Contains the parent abstract class as well as a class named TestHash which is based on this webpage [16]. Currently all algorithms use the same hash function but in the future a random variable will be introduced to this class to increase the domain of exploration. All hash subclasses must have a transform function that receives a string castable input and returns a value from 0 to 1. This is very important for the random sampling algorithm to work.

7.8 Singleton

This class contains all the parameters from the rest of the classes that might need to be saved in a configuration file. It's truly implemented as a singleton meaning that importing this class will always import the same instance.

7.9 ConsoleParser

This class is the connection between the Controller and the View. It parses user input.

8 Implementation of the Model

8.1 File

The parent class automatically creates the right type of File based on the extension of the name of the file. For Numpy Files I use the '.h5' extension and for json files I use the '.json' extension.

8.1.1 NumpyFile

After doing some research on Stack Overflow posts and Big Data forums as well as other resources I found a library oriented at big data called h5py [1] that is based on the Numpy library which is one of the most efficient python libraries for working with N-Dimensional arrays. After a lot of work, most of which was discarded I managed to create this class. It allows the user to store a depth one dictionary containing string keys and numpy ndarrays as values in a file. The good thing is that it's programmed in such a way that all changes are appended to the currently existing arrays within the h5 file called 'Databases'. This means that the size of the files can progressively grow based on the amount of data we dump into them. This files also allow for loseless compression of data using gzip and contain binary data. I also incorporated data type checking and array compression meaning that files containing 45,000,000 points weight less than 2MB. [10][8]

8.1.2 JsonFile

This type of files are useful for storing dictionaries of any depth. This class is programmed with the same mindset as the NumpyFile class meaning that all changes are appended to the file. If a key already exists in a file, the value of that key will be overwritten and if it doesn't exist it will be created. This class will be useful for storing configuration files and even python objects in a human readable way.

8.2 Filesystem

This class is a layer that interacts between python variables and files. It automatically creates the right type of file and manages the instances of already created Files. Another interesting characteristic is that this class is also implemented as a true singleton meaning that only one instance of the Filesystem will exist.

9 Implementation of the View

I initially considered different approaches for implementing the GUI:

1. Dash: it is a high level web GUI for Data Analytics. It looks really good by default
2. PyQt / PySide: it's a library that allows the creation of desktop apps based on Qt. I've implemented such interfaces before and the result works but it doesn't look really good. The positive thing about Qt is that it has been the standard GUI application for many years and a lot of documentation exists and the library is well maintained.
3. Kivy: similar to Qt, it offers a desktop interface but since it's a newer technology much less documentation exists and since it's open source bugs might be an issue.

After exploring all of the approaches I considered that the best option to save time would be to create a console based GUI. This approach has the advantage that integrating the app with testing scripts or communicating it with other scripts is really easy. It also means that if in the future a real GUI needs to be implemented, this functionality would still be useful to automate some processes for example using pipes.

9.1 Menu

This class implements a modular menu of any depth where options can either be folders or nodes. If the user selects a folder, the contents of that folder are displayed. If the user selects a node, the function associated to that node is executed. The selection is done using integers that are filtered to prevent a wrong input from crashing the app. There are only 3 nodes in the Menu. This means that only 3 functions can be executed by the user:

- Draw all '*.h5' files from the Database folder.
- Select a new selection_config.json file. Before chosing this option, you should place a valid certificate / selector json at the folder Model/Database/.Config/Eater_Execution. Once you select this option the file will be moved to Model/Database/.Config/Used_Eater_Execution.
- Start experimenting using the selection_config.json file to instantiate all objects as specified.

9.2 DatabaseDrawer

This class iterates over all the '.h5' files on the whole Database and generates a '.svg'. If a picture already exists it overwrites it. It has hardcoded properties which will be the only ones that can be displayed on the plot. I chose this file output since you can zoom into interesting areas of the plot without losing resolution.

9.3 Plotting

This class can visualize data from a NumpyFile. It does so by opening the data, processing the data (if derived properties are needed) and then saving it on the Database folder as a '.svg'.

9.3.1 Histogram

This class is used to represent the F test results in order to validate or reject the null hypothesis based on the *P Value* obtained when comparing the variance of two independent random variables.

9.3.2 Scatter

This class is used to plot the properties of the execution in a scatter plot.

10 Implementation of Utils

This package contains functions and other useful functionalities with a broader scope that could be used from View, Model or Controller packages. For example the class serialization is defined in this package since any class could need serialization at some point.

11 Hypothesis Testing

According to [12] at page 397, if the error probability of the test $\in [0, \frac{1}{2}]$, then you can accpet the null hypothesis or otherwise reject it. In the case of [10] at page 461, I created the following subsections in order to accept or reject the null hypothesis

11.1 Null hypotesis

H_0: The Affirmative Sampling Buffered and the Affirmative Sampling Windowed algorithms have a similar subset size given both algorithms have the same window and default subset sizes. The subset of unique ID's has a similar number of elements and also a similar variance.

In order to test the null hypothesis two plots will be generated. The first one will be a scatter plot in order to observe the evolution of the subsets of ASB and ASW over time. It also includes the average size of the subset in order to see if it converges to a constant value. Furthermore a plot of the intersection of the subsets between ASB and ASW is included as well as it's average in order to see if the intersection converges to a certain value as well. According to the classical theory of large numbers [17] [4], the sample mean of n independent and identically distributed d-dimensional random vectors with a certain mean and variance will converge to the population mean and to a variance distribution. The first plot will prove the convergence to a population mean and the histogram will prove that the variances of the two random variables can be explained by an F Distribution. If one of the epoch evaluations fails to pass the histogram test, it can always be argued that not enough values were sampled to observe convergence. If this null hypothesis is accepted, it means that the conjecture could be true but since we cannot explore all the domain since it's infinite, it will remain as a conjecture. Only if the similarity was proven mathematically, then it would become a theory.

11.2 Alternative Hypothesis

H_A It would be the opposite of the null hypothesis.

11.3 Data Collection

All tests and their respective certificates can be found at 20.2 and 20.3 so I won't be including them in the subsection.

11.4 Accept or reject hypothesis

From observing all the figures, the ones from *test 4* are the most representative 22, 23. They pass the F-distribution test and have been obtained with an experiment that has 100000 words and the more words, the better the result according to the theory of large numbers. We can observe that the F-statistic is close to 1 in all the figures. In some of the tests, the p value doesn't pass the test but this could be due to the fact that we are using a small number of words and also because while the window is not full, the deviation from the population mean is much higher which means that the first "window size" outliers should be discarded when computing the f-distribution. On the other hand, all the scatter plots seem to converge to some population means further accepting the null hypothesis. Therefor since there is mean convergence in the scatter plots and in the histograms, pvalue $\geq \alpha$ **??** (in the experiment that has the largest ammount of values), we can accept the null hypothesis. Furthermore if we observe the median vertical lines and the rest of the percentile lines, we can observe that both variances seem to have a very similar distribution of densities.

12 Workflow methodology

It's hard to make an accurate assessment on the time requirements for this project due to the innovative nature it has.

12.1 Version control

For version control I will use Github. This platform allows the creation of multiple branches. This branches will be organized from more criticallity to less. The more critical branches such as the release one need to be stable and the less critical branches will be the ones where development takes place. This type of project organization will allow parallel development of multiple features and prevent bugs from getting to the release branch.

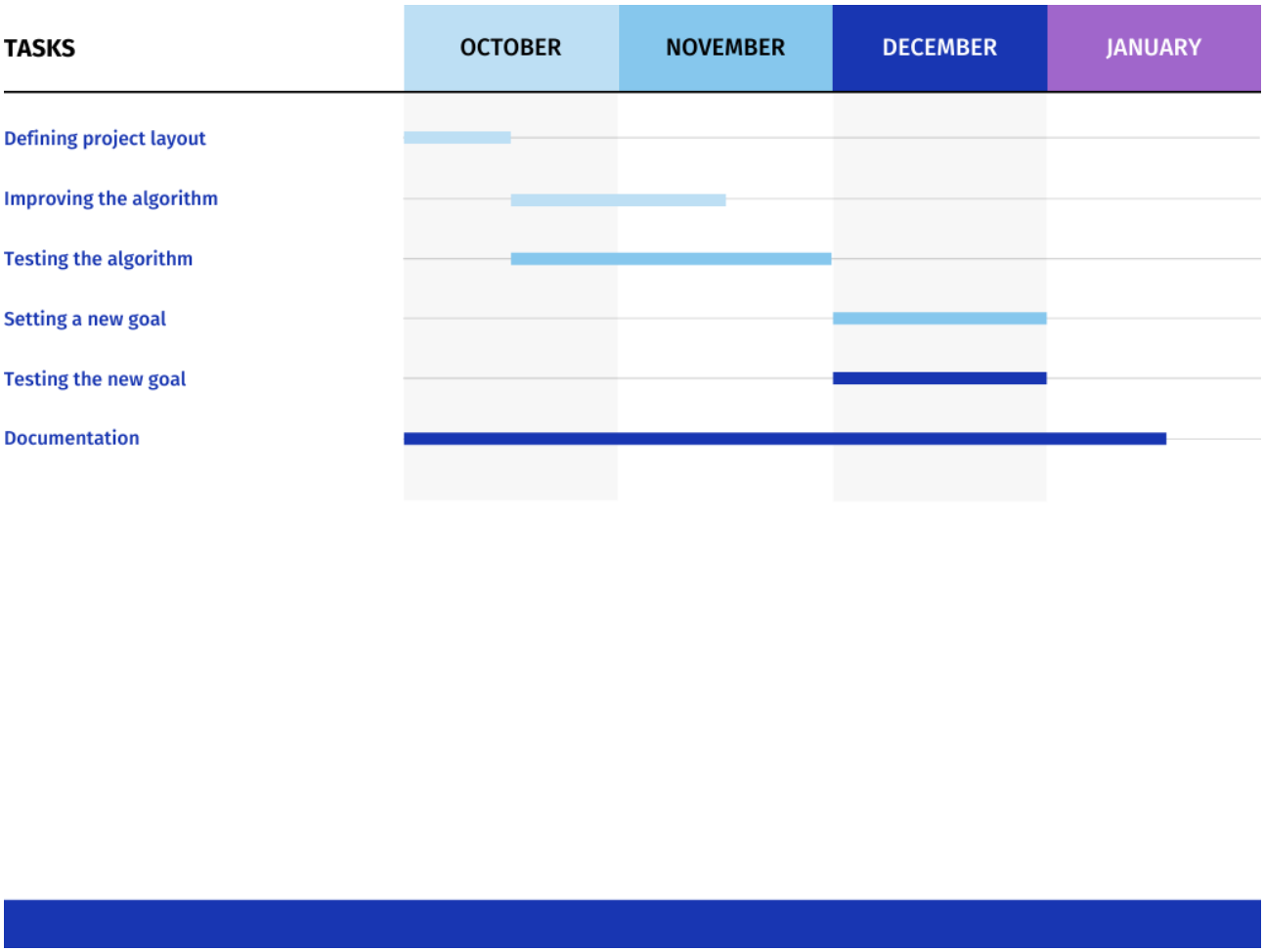
12.2 Objective monitoring

To monitor the objectives we will use an online *Kanban Board* manager hosted also on the Github plataform.

13 Gantt diagram

Gantt diagrams are an essential part of any project since they provide a way of predicting the future. Nevertheless, this project falls into the cathegory of I+D meaning that planning ahead is much more difficult since you can't really predict how long it will take to create and test something that doesn't exist and whether it will be successful or a disaster. This is why the following figure 2 is a synthesis of all the subtasks defined in the *Tasks* section. The diagram is purposely created in such a minimalistic way to showcase the uncertainty that still hovers over this project. Pretending to estimate the cost of any of the subtasks is not a realistic approach when it comes to I+D. I could do it and it would look really fancy but it certainly wouldn't be more accurate. What I mean by this is that defining more tasks than the necessary ones when all the tasks take an uncertain ammount of time will statistically cause more error since there are more points of failure. Instead of doing that, I opted for defining goals that will be fullfilled not because the prediction was accurate but because I decided to do so in the Gantt diagram. The dependencies between tasks in this project are very few meaning that for the most part this project will be pretty linear if we consider the blocks of tasks such as the project layout. However, within each block of tasks, the workflow methodology will be iterative meaning that implementation and testing will be intertwined. The times given to each of the tasks, more than a prediction will be a personal goal meaning that once the deadline is reached, the iterative implementation and testing cycle of that block will end.

Figure 2: Gantt diagram



13.1 Task time consumption

		Time (hours)
Defining project layout	Algorithms	8
	Hash functions	8
	Properties	16
	Data stream generators	8
	Input parser	4
	Output formatting	4
	Subtotal:	48
Improving the algorithm	Optimizing data structures	20
	Dynamic subsets	20
	Subtotal:	40
Testing the algorithm	Previous algorithms	8
	Required properties	8
	Required generators	8
	Optimize testing	8
	Extract conclusions	20
	Subtotal:	52
New goal for the project		80
	Subtotal:	80
Documenting		
	Subtotal:	80
Meetings with the tutor		15
	Subtotal:	15
Total:		315

14 Contingency planning

I included some contingency planning in the previous section at the *Setting a new goal* block since it will require a lot of planning to decide what path I follow once I reach that point depending on the results. In the tasks section I already described some of the paths I could take as this point of the project will be based upon the results of an algorithm that hasn't been fully developed yet so the uncertainty will be really high.

15 Sustainsability matrix

The following table comes from the project analysis described in the next sections. It's a tool that summarizes the environmental, economical and social impacts the development, deployment and maintenance our project will have. The sum of all the indexes will be 55 i a range that could go from -60 to 90. This means our project

Table 1: Sustainibility matrix

	Deployment	**Lifetime expectancy**	**Risks**
Environmental	15	15	0
Economical	15	15	0
Social	0	0	-5
Susteniability rank	30	30	-5

is sustainable.

16 Project deployment

The focus of this project will be about improving an algorithm. Some benchmarking tools will be developed in order to automate some of the tests and to quantify the improvements and extract some properties of the new algorithm. This will not be a commercial product since it would have to be integrated into another project. The production of something commercial is not in the scope of this project.

16.1 Environmental costs

This project will have the impact of one developer (me) and one laptop DELL XPS 13 which consumes 7-10 Watts. According to the FIB, this project is worth 18 university credits. Each credit is worth 25 hours. This means that the total number of hours that this project will take is 450. $450h \times 8.5W = 3825Wh$.

16.2 Economical costs

First of all I want to highlight the fact that I will treat this project as nonprofit endeavour. This means that the only benefits will be social and environmental at most but I won't focus on the economical profits that could be made.

16.2.1 Human costs

This project will be fully developed by me with the help of my teacher. If we suppose this was a real job and I was getting paid the minimal internship prices set by the university, then I would be earning 10 €/h. My teacher, due to his expertise and specialization in algorithms would be earning 60 €/h if he did private consulting. If we assume my Gantt diagram is correct and the cost of the tasks is also correct, the total time to develop this project would be 315 hours for my job and 15h for the teachers job. $315€ \times 10\frac{€}{h} + 15 \times 60\frac{€}{h} = 3150€ + 900€ = 4050€$

16.2.2 Material costs

In this section I will describe the costs associated to software and hardware. This costs will include the amortization of the purchase of all the material needed. The only material that will be needed in order to develop this project will be the office material in order for me to develop this project. This will include a light, chair, keyboard, mouse and laptop and extra screens. This is summarized in table 2.

Table 2: Hardware costs

	Price
Laptop	900€
Keyboard	20€
Mouse	20€
Light	25€
Screen	150€
Chair	150€
Office material	50€
Office material	50€
Total	1315€

Some of the other costs would be the software used. Because I'm a student I have many licenses free such as GitHub Pro, PyCharm Pro and GitKraken. If I were to develop this project on a professional setting I would have to pay all those licenses. This is summarized in table 3.

Table 3: Software costs

	Price (supposing 4 months of development)
GitKraken	28€
PyCharm	199€
GitHub	16€
Total	243€

16.2.3 Generic

This means that the only cost this project will have is the cost of the electricity consumed by me at home while working. If we take into account the latest average price in €for the MWh in Spain, we would realize it's 166€/MWh [15]. $166€/10^6 Wh = 0.000166€/Wh$. This means the laptop will consume a total of $3825Wh \times 0.000166€/Wh = 0.64€$. We would have to add to this value the cost of the light. I have 3 LED lights at my room which consume 10W each. This means that if I had the lights open during all my work time I would consume $450 \times 30 \times 0.000166 = 2.241€$. The total cost of electricity would be $2.241 + 0.64 = 2.881€$.

16.3 Social costs

The algorithm I will be improving has already been invented and has had a major social impact on our lives. All the big data companies must use random samplers in order to be able to reduce memory and time costs. Despite of the reduced cost that could be argued to have a positive impact on the environment, this algorithms can be used to monitor sensitive data and extract sensitive information. This can be used to censor information, predict election outcomes, categorize public opinion, personalize advertisements and many other forms of privacy intrusions. On the other hand it can be used to recommend better information, detect viruses, accelerate queries to databases, prosecute criminals, etc. In short, as all powerful things, it could be used for good and for evil. Since the algorithm is already being used by big data companies, this project will not have any impact in the sense that it won't bring anything new to the table.

17 Product lifetime

In this section I will evaluate the costs associated to the maintenance of this project.

17.1 Economical costs

The main costs will be the amortization of the material and software. If we suppose all the material will have an amortization period of 5 years this means that each year we will pay an extra cost of $\frac{1}{5} \times 1315€ = 270\frac{€}{\text{year}}$. Apart from the amortization of the materials there is also the maintenance of the software as it's illustrated in the following table 4.

Table 4: Software yearly cost

	Price per year
GitKraken	108 €
PyCharm	199 €
GitHub	48 €
Total	355€

17.2 Environmental costs

This project will be stored in the cloud consuming power in a remote GitHub server. I don't know the environmental cost of this remote server but I will assume it will be proportional to the monthly fee of a subscription to GitHub. If we consider that the $4\frac{€}{\text{month}}$ of the subscription is integrally going towards paying electricity bills, then this would mean that the monthly cost would be equivalent to $4€ \times \frac{1MWh}{166€} = 0.0241MWh$ [15]. Then if we assumed all that electricity is produced using coal which is the worst case scenario in terms of CO2 emissions per MWh [9], $0.0241MWh \times \frac{403.2kgCO_2}{1MWh} = 9.72\frac{kgCO_2}{month}$. To put that number into perspective that would be the equivalent of driving a standard car consuming 0.2kg/km [14] for 48km. This means that this project would consume less than me coming just one day to the university monthly. And all of this is assuming the worst case scenario which is probably orders of magnitude worse than the real figure.

On the other hand, if this project works and an increase in performance is achieved, it would reduce the time requirements of many algorithms meaning it would have a really huge impact on energy consumption.

17.3 Social costs

Even if I can improve the algorithm in a substantial way, the impact it will have in the big data industry will be minimal since very similar algorithms have already been deployed.

18 Risks

This project has absolutely no risk of having social or environmental consequences. At worst it will have no impact, at best it will reduce power consumption of the algorithms. On the other hand, developing the new algorithm can prove to be more difficult than expected and delay the completion of the whole project. Nevertheless, because of the simplicity of the project the costs of the delay can be extrapolated from the tables 2 and 4 and the sections 16.2.3 and 16.2.1.

19 Conclusion

The results obtained fulfill the goal of this project. Strong evidence was obtained proving the null hypothesis. Furthermore, the implications if this conjecture were true would be significant since the ASW algorithm is orders of magnitude faster than the mathematically proven ASB. On the other hand, due to the magnitude of this project and the fact that I work half time, I lacked more time to develop some ideas that I had. Specially the serialization, and filesystem implementations took me close to one month of development and I don't think the results can showcase how much effort I put into the architecture of the code. Finally, I'm happy by the obtained results and I believe I produced some quality software that has a lot of features making it really reusable, robust and scalable.

20 Annex

20.1 Code Implementation

20.1.1 Controller/Experiment/experiment.py

```
import time
import os

from Controller.Experiment.Evaluator.evaluator import Evaluator
from Controller.Experiment.Epoch.epochs import Epochs
from Model.Filesystem import Filesystem
from Utils.serializableClass import SerializableClass
from Utils.serializableClassAttribute import SerializableClassAttribute


class Experiment(SerializableClass):
    _epochs = SerializableClassAttribute(is_serializable_class=Epochs)
    _evaluator_parameters = SerializableClassAttribute(is_serializable_class=Evaluator)
    print("--------------------------Initialized Experiment")

    def __new__(cls):
        cls._config_dict = cls.serialized_class()
        return super().__new__(cls)

    def __init__(self):
        self._source_json = None
        self._update_source_json()
        self._selection_json = None
        self._load_selection_json()
        self._input_params = None
        self._load_input_params()

    def _update_source_json(self):
        self._source_json = Filesystem.source_json
        self._source_json.save(self._config_dict)

    def _load_selection_json(self):
        self._selection_json = Filesystem.selection_json

    def _load_input_params(self):

        source_json = self._source_json.load()

        selection_json = self._selection_json.load()

        merged_dict = self.merge_default_values_with_selected_values(source_json, selection_json)
        print("MERGED DICT: ", merged_dict)

        self._input_params = merged_dict

    def _load_subclass_params(self):
        raise NotImplementedError

    def merge_builder_catalog_node_with_selected(self, catalog_dict: dict,
                                                 selection_dict: list) -> list:
        ret = []
        if len(selection_dict) == 0:
            class_name = list(catalog_dict.keys())[0]
            recursive_content = self.merge_default_values_with_selected_values(catalog_dict[class_name], {})
            ret.append([class_name, recursive_content])
        else:
            if isinstance(selection_dict, list):
                for class_name, class_params in selection_dict:
                    recursive_content = self.merge_default_values_with_selected_values(catalog_dict[class_name],
                                                                                       class_params)
                    ret.append([class_name, recursive_content])
            else:
                raise Exception("If it's not none it MUST be a list")
        return ret

    def merge_default_values_with_selected_values(self, catalog_dict: dict, selection_dict) -> dict:
        def is_builder_catalog_node(current_dict):
            if isinstance(current_dict, dict):
                if '_builder_type' in current_dict.keys():
                    return True
```

```
                else:
                    return False
            else:
                return False
        ret = {}
        for key, item in catalog_dict.items():
            if not is_builder_catalog_node(item):
                if isinstance(item, dict):
                    next_selection_dict = {}
                    try:
                        next_selection_dict = selection_dict[key]
                    except KeyError:
                        pass
                    ret[key] = self.merge_default_values_with_selected_values(item, next_selection_dict)

                else: # it's a value
                    value = item
                    try:
                        value = selection_dict[key]
                    except KeyError:
                        pass
                    ret[key] = value
            else:
                next_selection_dict = {}
                try:
                    next_selection_dict = selection_dict[key]
                except KeyError:
                    pass
                ret[key] = self.merge_builder_catalog_node_with_selected(item, next_selection_dict)

        return ret

    def catalog_to_selection_dict(self, original_dict, epoch_dict):
        if isinstance(original_dict, dict):
            ret = {}
            for key, item in original_dict.items():
                if key in epoch_dict.keys():
                    ret[key] = self.catalog_to_selection_dict(original_dict[key], epoch_dict[key])
                else:
                    ret[key] = original_dict[key]
            return ret
        elif isinstance(original_dict, list):
            aux_value = []
            for class_name, class_params in original_dict:
                if class_name in epoch_dict.keys():
                    aux_value.append([class_name, self.catalog_to_selection_dict(class_params, epoch_dict[class_name])])
                else:
                    aux_value.append([class_name, class_params])
            return aux_value

        else:
            return epoch_dict

    def __call__(self):
        self._epochs = Epochs(**self._input_params['_epochs'])
        folder = str(time.asctime())
        for status in self._epochs:
            epoch_json = Epochs.status_to_json(status)
            merged_json = self.catalog_to_selection_dict(self._input_params, epoch_json)
            evaluator = Evaluator(**merged_json['_evaluator_parameters'], _certificate=merged_json)
            evaluator.set_output_file_name(folder + os.sep + self._epochs.name)
            evaluator.start()

    @staticmethod
    def eat():
        Filesystem.experiment_eater
```

20.1.2 Controller/Experiment/Epoch/epochs.py

```
import re

from Controller.Experiment.Epoch.EpochIter.epochIter import EpochIter
from Utils.serializableBuilderCatalog import SerializableBuilderCatalog
from Utils.serializableClass import SerializableClass
```

```
class Epochs(SerializableClass):
    _generator_factory = SerializableBuilderCatalog(
        EpochIter,
        builder_type='multiple'
    )

    def __init__(self, _generator_factory):
        self._generator_factory = [EpochIter(**class_pars) for class_name, class_pars in _generator_factory]
        self._generator_factory_inv = self._generator_factory[::-1]
        self._generators_inv = [g() for g in self._generator_factory_inv]
        self._status = self._init_status()
        self._n_path_to_value = [gf.path_to_value for gf in self._generator_factory]
        self._epochs_name = [0] * len(self._generator_factory)
        self._epochs_name[-1] = -1

    def _init_status(self):
        ret = []
        for i, g in enumerate(self._generators_inv):
            if i != 0:
                ret.append(next(g))
            else:
                ret.append(None)
        return ret[::-1]

    def __iter__(self):
        return self

    def __next__(self):
        """
        Tries to grab an item from a generator starting from the leftmost one,
        if the leftmost one is empty, it resets the empty generator (calling the generator
        factory to get a fresh generator.
        :return: A dictionary that represents the tree structure from all the EpochIterators at a certain epoch,
        this structure modifies the user selected execution and will be used to serialize the execution to create
        a certificate.
        """
        for inv_idx, generator in enumerate(self._generators_inv):

            # This consumes items from  the generator
            for element in generator:
                # If the generator is empty it doesn't enter this code!
                normal_idx = len(self._generators_inv) - inv_idx - 1
                self._status[normal_idx] = element
                self._epochs_name[normal_idx] += 1

                for aux_inv_idx, gf in enumerate(self._generator_factory_inv):
                    if aux_inv_idx < inv_idx:
                        aux_normal_idx = len(self._generators_inv) - inv_idx
                        self._generators_inv[aux_inv_idx] = gf()
                        self._status[aux_normal_idx] = next(self._generators_inv[aux_inv_idx])
                        self._epochs_name[aux_normal_idx] = 0

                # Only returns when it finds a non-empty generator
                return [tuple_object for tuple_object in zip(self._n_path_to_value, self._status)]
        raise StopIteration

    @staticmethod
    def status_to_json(status) -> dict:
        def recursive(dict_object, keys_path, final_value):
            key = keys_path[0]
            if len(keys_path) > 1:
                dict_object.setdefault(key, {})
                recursive(dict_object[key], keys_path[1:], final_value)
            elif len(keys_path) == 1:
                dict_object[key] = final_value

        ret = {}
        for path_to_value, value in status:
            keys = re.split(r'/', path_to_value)[1:]
            recursive(ret, keys, value)
        return ret

    @property
    def name(self):
        ret = ''
        for item in self._epochs_name:
```

```
            ret += f"{item}_"
        return ret[:-1]

    @staticmethod
    def __test__(array):
        e = Epochs(array)
        for status in e:
            print(status)

# Epochs.__test__([lambda: iter(range(10)), lambda: iter(range(5)), lambda: iter(range(5))])
```

20.1.3 Controller/Experiment/Epoch/EpochIter/epochIter.py

```
import numpy as np

from Utils.ValueDomain.valueDomain import ValueDomain
from Utils.serializableClass import SerializableClass
from Utils.serializableClassAttribute import SerializableClassAttribute
from Utils.path_in_dict import path_in_dict
from Model.Filesystem import Filesystem

FUNCTION_TYPES = {
    'range': lambda *params: iter(range(*params)),
    'linspace': lambda *params: iter(np.linspace(*params))
}


class EpochIter(SerializableClass):
    _path_to_value = SerializableClassAttribute(
        is_value=ValueDomain(
            value_type=str,
            bounds=[lambda x: path_in_dict(Filesystem.source_json.load(), x)],
            default=None
        )
    )
    _function_key = SerializableClassAttribute(
        is_value=ValueDomain(
            value_type=str,
            bounds=[lambda x: x in FUNCTION_TYPES],
            default='range'
        )
    )
    _function_params = SerializableClassAttribute(
        is_value=ValueDomain(
            value_type=tuple,
            bounds=[],
            default=(1,)
        )
    )

    def __init__(self, _path_to_value, _function_key, _function_params):
        self._path_to_value = _path_to_value
        self._function_key = _function_key
        self._function_params = _function_params

    def __call__(self):
        return FUNCTION_TYPES[self._function_key](*self._function_params)

    @property
    def path_to_value(self):
        return self._path_to_value
```

20.1.4 Controller/Experiment/Evaluator/evaluator.py

```
import numpy as np
import os

from Controller.Experiment.Evaluator.Generator.generatorBuilder import GeneratorBuilder
from Controller.Experiment.Evaluator.Property.property import Property
from Controller.Experiment.Evaluator.Algorithm.algorithmBuilder import AlgorithmBuilder
from Controller.Singleton import Singleton
from Model.Filesystem import Filesystem
from Utils.serializableClass import SerializableClass
from Utils.serializableBuilderCatalog import SerializableBuilderCatalog
from Utils.serializableClassAttribute import SerializableClassAttribute
from Utils.ValueDomain.valueDomain import ValueDomain
```

```
from Utils.lcm import lcm


class Evaluator(SerializableClass):
    _generator = SerializableClassAttribute(is_serializable_class=GeneratorBuilder)

    _algorithms = SerializableClassAttribute(is_serializable_class=AlgorithmBuilder)

    _properties = SerializableBuilderCatalog(
        Property,
        builder_type='multiple'
    )

    _n_total_words = SerializableClassAttribute(
        is_value=ValueDomain(
            value_type=int,
            bounds=[lambda x: 1 <= x <= 100000],
            default=int(10000)
        )
    )
    print("--------------------------Initialized Evaluator")

    def __init__(self,
                 *,
                 _generator,
                 _algorithms,
                 _n_total_words: int,
                 _properties,
                 _certificate
                 ):

        self._generator = \
            [GeneratorBuilder(class_name)(**class_params) for class_name, class_params in
             _generator['_builder_catalog']][0]

        self._algorithms = \
            [AlgorithmBuilder(class_name)(**class_params) for class_name, class_params in
             _algorithms['_builder_catalog']]
        self._n_total_words = _n_total_words
        self._properties = [Property(**params, algorithms=self._algorithms) for _, params in _properties]
        self._certificate = _certificate

        self._file_name = None

        self._elapsed_words = 0  # controls saving and buffer sizes
        self._intersection_sum = 0

        self._buffer_lcm = lcm([prop.sampling_frequency for prop in self._properties])
        self._buffer_size_properties = self.__get_buffer_size_properties()

        self._ndarray_dict = self.__get_ndarray_dict()
        self._idx = None
        self._output_certificate_name = None

    def start(self):
        print("start")
        for time_instant, generator_item in enumerate(self._generator):
            self._idx = time_instant
            self.__feed_all_algorithms(time_instant, generator_item)
            self.__update_ndarray_dict(ndarray_dict=self._ndarray_dict,
                                       properties=self._properties,
                                       time_instant=time_instant,
                                       buffer_size_properties=self._buffer_size_properties)
            self.__check_buffer_flush(time_instant)
            if self.__is_stop_condition(time_instant, self._n_total_words, self._buffer_size_properties):
                break
        print("Evaluator ended")

    def set_output_file_name(self, output_file_name):
        self._file_name = output_file_name
        path, file_name = os.path.split(self._file_name)
        self._output_certificate_name = path + os.sep + ".Certificates"+os.sep+file_name+"_cert.json"
        file = Filesystem[self._output_certificate_name]
        file.save(self._certificate)

    @staticmethod
```

```
def __update_ndarray_dict(ndarray_dict, properties, time_instant, buffer_size_properties):
    for prop in properties:
        if time_instant % prop.sampling_frequency == 0:
            property_function_result = prop.function
            chunk_idx = (time_instant % buffer_size_properties) // prop.sampling_frequency
            for idx1, val in enumerate(property_function_result):
                aux_idx = str(prop.which_algorithms[idx1])
                if isinstance(prop.which_algorithms[idx1], list):
                    aux_idx = ''.join([str(aux_idx) + '-' for aux_idx in prop.which_algorithms[idx1]])[:-1]
                ndarray_dict[prop.function_key][aux_idx][chunk_idx] = property_function_result[idx1]

@staticmethod
def __is_stop_condition(time_instant, n_total_words, buffer_size_properties) -> bool:
    return time_instant > n_total_words or buffer_size_properties == 0

def __check_buffer_flush(self, time_instant):
    if (time_instant + 1) % self._buffer_size_properties == 0:
        self.__dump_to_file()
        self._ndarray_dict = self.__get_ndarray_dict()

def __feed_all_algorithms(self, time_instant, generator_item):
    for a in self._algorithms:
        a.feed(item=generator_item, instant=time_instant)

def __dump_to_file(self):
    h5_filename = self._file_name+".h5"
    print("Saving buffer to file ----> ", h5_filename)
    numpy_file = Filesystem[h5_filename]
    print("NDARRAY DICT: ", self._ndarray_dict)
    numpy_file += self._ndarray_dict
    self._elapsed_words += self._buffer_size_properties
    self._buffer_size_properties = self.__get_buffer_size_properties()

def __get_buffer_size_properties(self):
    ret = min(
        (Singleton['BUFFER_SIZE_PROPERTIES'] // self._buffer_lcm) * self._buffer_lcm,
        ((self._n_total_words - self._elapsed_words) // self._buffer_lcm) * self._buffer_lcm
    )
    print(
        f"Singleton: {Singleton['BUFFER_SIZE_PROPERTIES']} // {self._buffer_lcm}) * {self._buffer_lcm} = {(Singleton['BUFFE
    print(
        f'(({self._n_total_words} - {self._elapsed_words}) // {self._buffer_lcm}) * {self._buffer_lcm} = {((self._n_total_w
    return ret

def __get_ndarray_dict(self) -> {str: np.ndarray}:
    ret = {}
    for prop in self._properties:
        for arr in enumerate(prop.which_algorithms):
            for idx, which_alg in enumerate(arr):
                numpy_1darray = np.ones(
                    shape=(int(self._buffer_size_properties / prop.sampling_frequency),),
                    dtype=prop.array_data_type
                ) * -1
                if isinstance(which_alg, int):
                    str_which_alg = str(which_alg)
                    try:
                        ret[prop.function_key]
                    except KeyError:
                        ret[prop.function_key] = {}
                    try:
                        ret[prop.function_key][str(str_which_alg)] = numpy_1darray
                    except KeyError:
                        raise Exception
                elif isinstance(which_alg, list):
                    tuple_to_string = ''.join([str(aux_idx) + '-' for aux_idx in which_alg])[:-1]
                    try:
                        ret[prop.function_key][tuple_to_string]

                    except KeyError:  # case dict not created yet
                        ret[prop.function_key] = {}
                    ret[prop.function_key][tuple_to_string] = numpy_1darray
                else:
                    raise Exception
    return ret
```

20.1.5 Controller/Experiment/Evaluator/Algorithm/algorithm.py

```
from abc import abstractmethod, ABC

from Controller.Experiment.Evaluator.Algorithm.Hash.hashBuilder import HashBuilder
# from Controller.Algorithm.algorithmBuilder import AlgorithmBuilder TODO: if this link is present then import cycle appears !!!
from Utils.serializableClassAttribute import SerializableClassAttribute
from Utils.serializableClass import SerializableClass


class Algorithm(ABC, SerializableClass):
    _hash_function = SerializableClassAttribute(is_serializable_class=HashBuilder)
    print('---------------------------Initialized Algorithm')

    def __init__(self,
                 *args,
                 _hash_function=None,
                 **kwargs):
        if _hash_function is not None:
            self._hash_function = \
                [HashBuilder(class_name)(**class_params) for class_name, class_params in
                 _hash_function["_builder_catalog"]][0]

    @abstractmethod
    def feed(self, item, instant):
        pass

    @abstractmethod
    def status(self, time_instant):
        pass
```

20.1.6 Controller/Experiment/Evaluator/Algorithm/affirmativeSamplingAlgorithm.py

```
import heapq

from Controller.Experiment.Evaluator.Algorithm.algorithm import Algorithm
from Controller.Experiment.Evaluator.Algorithm.sequenceElement import SequenceElement
from Utils.ValueDomain.valueDomain import ValueDomain
from Utils.check_all_supers import merge_dicts, serialize_supers
from Utils.serializableClassAttribute import SerializableClassAttribute


class AffirmativeSampling(Algorithm):
    _chosen_max_size = SerializableClassAttribute(
        is_value=ValueDomain(
            value_type=int,
            bounds=[lambda x: x >= 2, lambda x: x <= 100],
            default=2
        )
    )
    print("---------------------------Initialized AffirmativeSampling")

    def __init__(self, _chosen_max_size, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self._name = "AffirmativeSampling"
        # print("AffirmativeSampling initialized")
        self._subset = {}
        self._chosen = []
        self._expanded = []
        heapq.heapify(self._chosen)
        heapq.heapify(self._expanded)
        self._chosen_max_size = _chosen_max_size
        self._idx = -1

    def __repr__(self):
        ret = f'{self._name}'
        return ret

    def feed(self, item, instant):
        hash_value = self._hash_function.transform(item, verbose=False)
        self.__update(item, hash_value, instant)

    def __update(self, item: str, hash_value: float, instant=0):
        # while the number of distinct elements fits in the chosen set
        if len(self._chosen) < self._chosen_max_size:
            if hash_value not in self._subset:
```

```
                # I add those elements to the set if they don't exist
                element = SequenceElement(hash_value, item)
                self._subset.update({hash_value: element})
                heapq.heappush(self._chosen, element)
            else:
                # and if they exist I update the frequency
                self._subset[hash_value] += 1
        # once the chosen set is full
        else:
            # if the element is in the subset I update the frequency
            if hash_value in self._subset:
                self._subset[hash_value] += 1
            # If it doesn't exist
            else:
                # if the element hash improves the minimum hash of the chosen set
                # then we remove the minimum element from the chosen set and move it
                if hash_value > self._chosen[0].key:
                    element = SequenceElement(hash_value, item)
                    self._subset.update({hash_value: element})
                    deleted_element = heapq.heappop(self._chosen)  # Deletes minimum element from heapq
                    heapq.heappush(self._expanded, deleted_element)
                    heapq.heappush(self._chosen, element)
                else:
                    if len(self._expanded) > 0:
                        min_hash_value = self._expanded[0].key
                        if hash_value > min_hash_value:
                            element = SequenceElement(hash_value, item)
                            self._subset.pop(min_hash_value)
                            self._subset.update({hash_value: element})
                            heapq.heappush(self._expanded, element)
                            heapq.heappop(self._expanded)
        return None

    def status(self, instant):
        print(self._subset)

    @property
    def subset_size(self):
        return len(self._subset)

    @property
    def chosen_len(self):
        return len(self._chosen)

    @property
    def expanded_len(self):
        return len(self._expanded)

    @property
    def subset(self):
        return self._subset

    @classmethod
    def serialized_class(cls) -> dict:
        ret = super().serialized_class()
        ret = merge_dicts(ret, serialize_supers(cls))
        return ret
```

20.1.7 Controller/Experiment/Evaluator/Algorithm/affirmativeSamplingBufferedAlgorithm.py

```
from collections import deque

from Controller.Experiment.Evaluator.Algorithm.algorithm import Algorithm
from Controller.Experiment.Evaluator.Algorithm.affirmativeSamplingAlgorithm import AffirmativeSampling
from Utils.ValueDomain.valueDomain import ValueDomain
from Utils.serializableClassAttribute import SerializableClassAttribute


class AffirmativeSamplingBuffered(Algorithm):
    _affirmative_sampling_creation_params = SerializableClassAttribute(is_serializable_class=AffirmativeSampling)
    _buffer_size = SerializableClassAttribute(
        is_value=ValueDomain(
            value_type=int,
            bounds=[lambda x: 1 <= x <= 2000],
            default=int(1)
        )
```

```
    )
    print("--------------------------Initialized AffirmativeSamplingBuffered")

    def __init__(self, *args, _affirmative_sampling_creation_params, _buffer_size, **kwargs):
        super().__init__(*args, _hash_function=None, **kwargs)
        self._name = "AffirmativeSamplingBuffered"
        self._buffer_size = _buffer_size
        self._buffer_dummy = deque(maxlen=self._buffer_size)
        self._affirmative_sampling_creation_params = _affirmative_sampling_creation_params
        self._dummy_algorithm = None

    def feed(self, item, instant):
        self._dummy_algorithm = AffirmativeSampling(**self._affirmative_sampling_creation_params)
        self._buffer_dummy.append(item)
        for idx, s in enumerate(self._buffer_dummy):
            self._dummy_algorithm.feed(item=s, instant=instant)

    def status(self, instant):
        return self.subset

    def __repr__(self):
        return self._name

    @property
    def subset_size(self):
        ret = self._dummy_algorithm.subset_size
        return ret

    @property
    def chosen_len(self):
        return self._dummy_algorithm.chosen_len

    @property
    def expanded_len(self):
        return self._dummy_algorithm.expanded_len

    @property
    def subset(self):
        return self._dummy_algorithm.subset
```

20.1.8 Controller/Experiment/Evaluator/Algorithm/affirmativeSamplingWindowedAlgorithm.py

```
import heapq

from Controller.Experiment.Evaluator.Algorithm.affirmativeSamplingAlgorithm import AffirmativeSampling
from Controller.Experiment.Evaluator.Algorithm.sequenceTimerElement import TimerElement
from Utils.ValueDomain.valueDomain import ValueDomain
from Utils.check_all_supers import serialize_supers, merge_dicts
from Utils.serializableClassAttribute import SerializableClassAttribute


class AffirmativeSamplingWindowed(AffirmativeSampling):
    _window_size = SerializableClassAttribute(
        is_value=ValueDomain(
            value_type=int,
            bounds=[lambda x: 1 <= x <= 100000],
            default=int(1)
        )
    )
    print("--------------------------Initialized AffirmativeSamplingWindowed")

    def __init__(self,
                 *,
                 _window_size,
                 **kwargs):
        super().__init__(**kwargs)
        self._name = "AffirmativeSampling"
        # print("AffirmativeSamplingWindowed initialized")
        self._window_size = _window_size
        # Will contain the information about the real length of each subset considering we use
        # a lazy garbage collector to update the structures which is not consistent.
        self.__subset_len = 0
        self.__chosen_len = 0
        self.__expanded_len = 0
        self._instant = None
```

```
def __garbage_collector_subset(self, instant):
    """
    Removes all items from the dictionary that are not alive.
    Heap queue structures are not updated and have to be checked when the min element is popped
    until an alive element is found.
    :return: void / None
    """
    items = list(self._subset.items())
    for key, item in items:
        if item.is_dead(instant):
            # print("I did something useful")
            self._subset.pop(key)

def __garbage_collector_min_item(self, instant):
    """
    Remove minimum element from heapq until an alive minimum is found
    :param instant:
    :return:
    """
    # while the minimum item is dead, find a new minimum
    while len(self._chosen) > 0 and self._chosen[0].is_dead(instant):
        item = self._chosen.pop()  # and erase the dead minimum
        self._subset.pop(item.key)
        self.__subset_len -= 1
        self.__chosen_len -= 1

def __garbage_collector_chosen(self, instant):
    for idx, item in enumerate(self._chosen):
        if item.is_dead(instant):
            self._chosen.pop(idx)
            self.__chosen_len -= 1
            if self._subset.get(item.key):
                self._subset.pop(item.key)
                self.__subset_len -= 1

def __garbage_collector_expanded(self, instant):
    while self._expanded[0].is_dead(instant):
        self._expanded.pop(0)
        self.__expanded_len -= 1
        self.__subset_len -= 1

def feed(self, item, instant):
    hash_value = self._hash_function.transform(item, verbose=False)
    self.__update(item=item, hash_value=hash_value, instant=instant)

# @measure_time
def __update(self, item: str, hash_value: float, instant=0):
    # while the number of distinct elements fits in the chosen set
    self.__garbage_collector_chosen(instant)  # erase dead elements from chosen
    self._instant = instant
    if self.__chosen_len < self._chosen_max_size:
        if hash_value not in self._subset:
            # I add those elements to the set if they don't exist
            element = TimerElement(key=hash_value, value=item, death_timer=instant + self._window_size)
            self._subset.update({hash_value: element})
            heapq.heappush(self._chosen, element)
            self.__chosen_len += 1
            self.__subset_len += 1
        else:
            if self._subset[hash_value].is_alive(instant):
                # and if they exist I update the frequency
                self._subset[hash_value] += 1
            else:
                self._subset.pop(hash_value)
                self.__update(item, hash_value, instant)
    # once the chosen set is full
    else:
        # If the element is in the subset and is alive
        if hash_value in self._subset:
            # I update the frequency
            self._subset[hash_value] += 1

        # If it doesn't exist in the subset
        else:
            # if the element hash improves the minimum hash of the chosen set
            # then we remove the minimum element from the chosen set and move it
```

```
                if hash_value > self._chosen[0].key:
                    element = TimerElement(key=hash_value, value=item, death_timer=instant + self._window_size)
                    self._subset.update({hash_value: element})
                    deleted_element = heapq.heappop(self._chosen)  # Deletes minimum element from heapq
                    deleted_element.insert(0, instant + self._window_size)
                    heapq.heappush(self._expanded, deleted_element)
                    heapq.heappush(self._chosen, element)
                    self.__expanded_len += 1
                    # self.__chosen_len += 0
                else:
                    if len(self._expanded) > 0:
                        self.__garbage_collector_expanded(instant)
                        min_hash_value = self._expanded[0].key
                        if hash_value > min_hash_value or min_hash_value is None:
                            element = TimerElement(key=hash_value, value=item, death_timer=instant + self._window_size)
                            if self._subset.get(min_hash_value):
                                self._subset.pop(min_hash_value)  # check if it exists in subset
                            self._subset.update({hash_value: element})
                            heapq.heappush(self._expanded, element)
                            heapq.heappop(self._expanded)
                            # self.__expanded_len += 0
        return None

    def status(self, instant):
        print(self._subset)

    @property
    def window_size(self):
        return self._window_size

    @property
    def chosen_len(self):
        return self.__chosen_len

    @property
    def expanded_len(self):
        return self.__expanded_len

    @property
    def subset(self):
        self.__garbage_collector_subset(self._instant)
        return self._subset

    @property
    def subset_size(self):
        return len(self.subset)

    @classmethod
    def serialized_class(cls) -> dict:
        ret = super().serialized_class()
        ret = merge_dicts(ret, serialize_supers(cls))
        return ret
```

20.1.9 Controller/Experiment/Evaluator/Algorithm/algorithmBuilder.py

```
from Controller.Experiment.Evaluator.Algorithm.affirmativeSamplingAlgorithm import AffirmativeSampling
from Controller.Experiment.Evaluator.Algorithm.affirmativeSamplingBufferedAlgorithm import AffirmativeSamplingBuffered
from Controller.Experiment.Evaluator.Algorithm.affirmativeSamplingWindowedAlgorithm import AffirmativeSamplingWindowed
from Utils.serializableBuilderCatalog import SerializableBuilderCatalog
from Utils.serializableClass import SerializableClass


class AlgorithmBuilder(SerializableClass):
    _builder_catalog = SerializableBuilderCatalog(
        AffirmativeSamplingBuffered,
        AffirmativeSamplingWindowed,
        AffirmativeSampling,
        builder_type='multiple'
    )
    print("--------------------------Initialized AlgorithmBuilder")

    def __new__(cls, algorithm_name, *args, **kwargs):
        try:
            class_algorithm = cls._builder_catalog.subclasses[algorithm_name]
            return class_algorithm
        except KeyError:
```

```
            raise NotImplementedError
```

20.1.10 Controller/Experiment/Evaluator/Algorithm/sequenceElement.py

```
class SequenceElement:
    def __init__(self, key, value):
        self._key = key
        self._value = value
        self._count = 1

    @property
    def key(self):
        return self._key

    @property
    def value(self):
        return self._value

    @property
    def count(self):
        return self._count

    def __iadd__(self, other: int):
        self._count += other
        return self

    def __repr__(self):
        ret = f'{self._value}~x({self._count})'
        return ret

    def __lt__(self, other):
        return self.key < other.key
```

20.1.11 Controller/Experiment/Evaluator/Algorithm/sequenceTimerElement.py

```
from Controller.Experiment.Evaluator.Algorithm.sequenceElement import SequenceElement


class TimerElement(SequenceElement):
    def __init__(self, death_timer=None, *args, **kwargs):
        super().__init__(*args, **kwargs)
        assert death_timer is not None and death_timer > 0
        self.__death_timer = death_timer

    def is_dead(self, instant):
        if self.__death_timer <= instant:
            return True
        return False

    def is_alive(self, instant):
        return not self.is_dead(instant)

    def insert(self, num, instant):
        self._count += num
        self.__death_timer = instant
```

20.1.12 Controller/Experiment/Evaluator/Algorithm/Hash/hash.py

```
from abc import abstractmethod, ABC

from Utils.serializableClass import SerializableClass


class Hash(ABC, SerializableClass):
    print("--------------------------Initialized Hash")
    def __init__(self, *args, **kwargs):
        pass

    @abstractmethod
    def transform(self, item: str) -> float:
        raise NotImplementedError
```

20.1.13 Controller/Experiment/Evaluator/Algorithm/Hash/hashBuilder.py

```
from Utils.serializableClass import SerializableClass
from Utils.serializableBuilderCatalog import SerializableBuilderCatalog
from Controller.Experiment.Evaluator.Algorithm.Hash.testHash import TestHash
from Controller.Experiment.Evaluator.Algorithm.Hash.sha256 import Sha256


class HashBuilder(SerializableClass):
    _builder_catalog = SerializableBuilderCatalog(
        TestHash,
        Sha256
    )
    print("--------------------------Initialized HashBuilder")

    def __new__(cls, hash_name, *args, **kwargs):
        try:
            catalog_class = cls._builder_catalog.subclasses[hash_name]
            return catalog_class
        except KeyError:
            print("Hash name not available")
            raise Exception
        return None

    def __init__(self, *args, **kwargs):
        pass
```

20.1.14 Controller/Experiment/Evaluator/Algorithm/Hash/sha256.py

```
from abc import ABC
import hashlib

from Controller.Experiment.Evaluator.Algorithm.Hash.hash import Hash
from Utils.string_to_binary import string_to_binary


class Sha256(Hash, ABC):
    divide_value = 2 ** 256 - 1
    print("--------------------------Initialized Sha256")

    def transform(self, item: str, verbose=False) -> float:
        val_hex = hashlib.sha256(string_to_binary(item)).hexdigest()
        val_int = int(val_hex, 16)
        ret = val_int / self.divide_value
        assert ret <= 1
        if verbose:
            print(f'{item} -> {ret}')
        return ret
```

20.1.15 Controller/Experiment/Evaluator/Algorithm/Hash/testHash.py

```
from __future__ import annotations
import hashlib
import random

from Controller.Experiment.Evaluator.Algorithm.Hash.hash import Hash
from Utils.ValueDomain.valueDomain import ValueDomain
from Utils.serializableClassAttribute import SerializableClassAttribute


class TestHash(Hash):
    _seed = SerializableClassAttribute(
        is_value=ValueDomain(
            value_type=int,
            bounds=[lambda x: x >= 0],
            default=int(0)
        )
    )
    print("--------------------------Initialized TestHash")

    def __init__(self,
                 *,
                 _seed,
                 **kwargs):
        super().__init__(**kwargs)
        self._name = "TestHash"
```

```
        self._seed = _seed
        self._random_generator = random.Random(_seed)

    def transform(self, item, verbose=False):
        message_utf8 = str(item).encode('utf8')
        output = hashlib.md5(message_utf8).hexdigest() # transform string to hexadecimal
        output = int(output, 16) % 18327919
        output /= 18327919  # divide output by same prime number to get range [0,1] # 1 #
        if verbose:
            print(f'{item} -> {output}')
        return output
```

20.1.16 Controller/Experiment/Evaluator/Generator/generator.py

```
from abc import abstractmethod, ABC

from Utils.serializableClass import SerializableClass


class Generator(ABC, SerializableClass):

    def __init__(self, *args, **kwargs):
        self._generator_words = {}

    @property
    def subset(self):
        return self._generator_words

    @abstractmethod
    def __iter__(self) -> str:
        pass

    @property
    def generator_words(self):
        return self._generator_words
```

20.1.17 Controller/Experiment/Evaluator/Generator/generatorBuilder.py

```
from Controller.Experiment.Evaluator.Generator.textGenerator import TextGenerator
from Controller.Experiment.Evaluator.Generator.zipfGenerator import ZipfGenerator
from Utils.serializableBuilderCatalog import SerializableBuilderCatalog
from Utils.serializableClass import SerializableClass


class GeneratorBuilder(SerializableClass):
    _builder_catalog = SerializableBuilderCatalog(
        ZipfGenerator,
        TextGenerator
    )
    print("---------------------------Initialized GeneratorBuilder")

    def __new__(cls, generator_name):
        try:
            catalog_class = cls._builder_catalog.subclasses[generator_name]
            return catalog_class
        except KeyError:
            print("Generator name not available")
            raise Exception
        return None
```

20.1.18 Controller/Experiment/Evaluator/Generator/textGenerator.py

```
import re
import os

from Controller.Experiment.Evaluator.Generator.generator import Generator
from Utils.check_all_supers import merge_dicts, serialize_supers


class TextGenerator(Generator):
    print("--------------------------Initialized TextGenerator")
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self._name = "TestGenerator"
        generator_params = kwargs["generator_params"]
```

```
        kwargs.pop("generator_params")
        self.__init(**generator_params)

    def __init(self, database, *kwargs):
        self._database = database

    def __repr__(self):
        ret = f'I\'m {self._name} and I\'m reading from {os.path.basename(self._database)}'
        return ret

    def __iter__(self):
        chunk_size = 64
        f = open(self._database, 'r')
        while True:
            data = f.read(chunk_size)
            if not data:
                break
            data = re.split('[^a-zA-Z\']', data)
            data = list(filter(str.strip, data))
            for word in data:
                yield word
        f.close()

    @classmethod
    def serialized_class(cls) -> dict:
        ret = super().serialized_class()
        ret = merge_dicts(ret, serialize_supers(cls))
        return ret
```

20.1.19 Controller/Experiment/Evaluator/Generator/zipfGenerator.py

```
import random
import numpy as np

from Controller.Experiment.Evaluator.Generator.generator import Generator
from Utils.ValueDomain.valueDomain import ValueDomain
from Utils.check_all_supers import merge_dicts, serialize_supers
from Utils.serializableClassAttribute import SerializableClassAttribute


class ZipfGenerator(Generator):
    _ndistinct = SerializableClassAttribute(
        is_value=ValueDomain(
            value_type=int,
            bounds=[lambda x: x < 1000000],
            default=500000
        )
    )
    _beta = SerializableClassAttribute(
        is_value=ValueDomain(
            value_type=float,
            bounds=[lambda x: 1 <= x < 4],
            default=2.0
        )
    )
    _rank_offset_left = SerializableClassAttribute(
        is_value=ValueDomain(
            value_type=int,
            bounds=[lambda x: x < ZipfGenerator._ndistinct.value.default],
            default=1000
        )
    )
    _rank_offset_right = SerializableClassAttribute(
        is_value=ValueDomain(
            value_type=int,
            bounds=[lambda x: x < ZipfGenerator._ndistinct.value.default],
            default=0
        )
    )
    _seed = SerializableClassAttribute(
        is_value=ValueDomain(
            value_type=int,
            bounds=[lambda x: x >= 0],
            default=0
        )
    )
```

```
    print("--------------------------Initialized ZipfGenerator")

    def __init__(self,
                 *args,
                 _beta,
                 _ndistinct,
                 _rank_offset_left,
                 _rank_offset_right,
                 _seed,
                 **kwargs):
        super().__init__(*args, **kwargs)
        self._ndistinct = _ndistinct
        self._beta = _beta
        assert _rank_offset_left >= 0
        self._rank_offset_left = _rank_offset_left
        assert _rank_offset_right >= 0
        self._rank_offset_right = _rank_offset_right
        self.__number_parts = self.__get_number_parts()
        self.__probabilities = self.__get_probabilities()
        self._seed = _seed
        self._random_generator = random.Random(self._seed)

    def __repr__(self):
        ret = f'ZipfGenerator(beta: {self._beta}'
        return ret

    def __get_number_parts(self):
        """
        :return: number of parts of the cake (Harmonic sum)
        """
        sum = 0
        for i in range(self._rank_offset_left, self._ndistinct - self._rank_offset_right):
            sum += 1 / i ** self._beta
        return sum

    def __get_probabilities(self):
        res = []
        sum = 0
        for i in range(self._rank_offset_left, self._ndistinct - self._rank_offset_right):
            sum += 1 / ((i ** self._beta) * self.__number_parts)
            res.append(sum)
        return np.array(res)

    def __get_value(self):
        """
        bisection algorithm
            left  <--->  a[i-1] < v <= a[i]
        """
        index = np.searchsorted(self.__probabilities, self._random_generator.random(), side='left')
        return f'{index}'

    def __iter__(self):
        while True:
            value = self.__get_value()
            if value in self._generator_words:
                self._generator_words[value] += 1
            else:
                self._generator_words.update({value: 1})
            yield value

    @classmethod
    def serialized_class(cls) -> dict:
        ret = super().serialized_class()
        ret = merge_dicts(ret, serialize_supers(cls))
        return ret
```

20.1.20 Controller/Experiment/Evaluator/Property/property.py

```
from Utils.ValueDomain.valueDomain import ValueDomain
from Utils.serializableClass import SerializableClass
from Utils.serializableClassAttribute import SerializableClassAttribute
import numpy as np

FUNCTION_MIN_NUM_ARGS = {
    'subset size': 1,
```

```
    'intersection size': 2
}


def compare_intersection(algs):
    for tup in algs:
        distinct_words = set(algs[0][0].subset.keys())
        for alg in tup:
            distinct_words = distinct_words.intersection(set(alg.subset.keys()))
        ret = distinct_words
    return ret


FUNCTION_TYPES = {
    'subset size': lambda *args: [a.subset_size for a in args],
    'intersection size': lambda *args: [len(compare_intersection(args)) for arg in args]
}

DATA_TYPES = {
    'int': np.int,
    'float': np.float
}


class Property(SerializableClass):
    _function_key = SerializableClassAttribute(
        is_value=ValueDomain(
            value_type=str,
            bounds=[lambda x: x in FUNCTION_TYPES],
            default='subset size'
        )
    )
    _data_type_key = SerializableClassAttribute(
        is_value=ValueDomain(
            value_type=str,
            bounds=[lambda x: x in DATA_TYPES],
            default='int'
        )
    )
    _sampling_frequency = SerializableClassAttribute(
        is_value=ValueDomain(
            value_type=int,
            bounds=[lambda x: x >= 1],
            default=1
        )
    )
    _which_algorithms = SerializableClassAttribute(
        is_value=ValueDomain(
            value_type=list,
            bounds=[],
            default=[0]
        )
    )

    def __init__(self, *,
                 _function_key,
                 _data_type_key,
                 _sampling_frequency,
                 _which_algorithms,
                 algorithms):
        self._function_key = _function_key
        self._data_type_key = _data_type_key
        self._sampling_frequency = _sampling_frequency
        self._algorithms = self._init_algorithms(_which_algorithms, algorithms)
        self._which_algorithms = _which_algorithms

    @staticmethod
    def _init_algorithms(which_algorithms, algorithms):
        """
        print("_init_algorithms", ret)
        _init_algorithms [[AffirmativeSamplingBuffered, AffirmativeSampling]]
        print(f"self._function_key:{self._function_key}, algorithms: {self._algorithms}", )
        self._function_key:subset size, algorithms: [AffirmativeSamplingBuffered, AffirmativeSampling]
        :param which_algorithms:
        :param algorithms:
        :return:
```

```
        """
        ret = None
        if isinstance(which_algorithms[0], int):
            return [algorithms[idx] for idx in which_algorithms]
        else:

            aux = []
            ret = aux
            for tup in which_algorithms:
                aux1 = []
                for idx in tup:
                    aux1.append(algorithms[idx])
                aux.append(aux1)
            return ret

    @property
    def function_key(self):
        return self._function_key

    @property
    def function(self):
        return FUNCTION_TYPES[self._function_key](*self._algorithms)

    @property
    def array_data_type(self):
        return DATA_TYPES[self._data_type_key]

    @property
    def sampling_frequency(self):
        return self._sampling_frequency

    @property
    def which_algorithms(self):
        return self._which_algorithms
```

20.1.21 Controller/Singleton/__init__.py

```
from Controller.Singleton.singleton import __Singleton
Singleton = __Singleton()
from .__init__ import Singleton
```

20.1.22 Controller/Singleton/singleton.py

```
import os

from root_dir import ROOT_DIR


class __Singleton(dict):
    def __init__(self):
        super().__init__()
        self['MAX_FILESYSTEM_MEMORY'] = '5G'

        self['DATABASE_PATH'] = f'{ROOT_DIR}{os.sep}Model{os.sep}Database'

        self['BUFFER_SIZE_PROPERTIES'] = 100000

        self['CONFIG_DIR'] = f'.Configuration'

        self['CURR_CONFIG_FILENAME'] = 'source_config.json'

        self['CURR_CONFIG_FILE'] = f'{self["CONFIG_DIR"]}{os.sep}{self["CURR_CONFIG_FILENAME"]}'

        self['CURR_SELECTION_FILENAME'] = f'selection_config.json'

        self['CURR_SELECTION_FILE'] = f'{self["CONFIG_DIR"]}{os.sep}{self["CURR_SELECTION_FILENAME"]}'

        self['EXECUTION_SELECTION_EATER_FOLDER'] = f'{self["CONFIG_DIR"]}{os.sep}Eater_Execution'

        self['USED_SELECTIONS_EXEC'] = f"{self['CONFIG_DIR']}{os.sep}Used_Eater_Execution"

        self['ALPHA_TEST'] = 0.05
```

20.1.23 Controller/UI/ConsoleParser/consoleParser.py

```
from View.DatabaseDrawer.databaseDrawer import DatabaseDrawer
from View.Menu.menu import Menu
from Controller.Experiment.experiment import Experiment


class ConsoleParser:
    def __init__(self, is_verbose):
        self._is_verbose = is_verbose
        self._menu = Menu(None)
        self._menu.add_submenu(name='Experiment')
        self._menu.add_option(name="Draw All Files in Database", action=lambda: DatabaseDrawer()())
        submenu = self._menu['Experiment']
        submenu.add_option(name='Load a New Selection', action=lambda: Experiment.eat())
        submenu.add_option(name='Start Experiment', action=lambda: Experiment()())

    def __repr__(self):
        return str(self._menu)

    @staticmethod
    def _footer():
        ret = "( Q ) ___ Go back"
        return ret

    def start(self):
        while not self._menu.quit:
            if self._is_verbose:
                print('++++++++++++++++++++++++++++++++++++')
                print(f':{self._menu.path}$')
                print()
                print(self)
                print(ConsoleParser._footer())
                print('------------------------------------')

            while True:
                a = input('>>> ')
                try:
                    self._menu.transition(a)
                except IndexError:
                    raise IndexError
                else:
                    # case input is valid
                    break
        self.quit()

    def quit(self):
        pass
```

20.1.24 Model/Filesystem/filesystem.py

```
import os
import re
import shutil
import numpy as np

from Controller.Singleton import Singleton
from Model.Filesystem.File.file import File


class __Filesystem:
    def __init__(self):
        self._all_files_in_database = {}
        self._init_all_files_in_database()

    def explore_folder_recursive(self, absolute_path) -> dict:
        ret = {}
        for root, _, files in os.walk(absolute_path):
            for file in files:
                relative_folder_path = re.split(self.database_path, root)[1]
                relative_file_path = relative_folder_path + os.sep + file
                if relative_file_path[0] == os.sep:
                    relative_file_path = relative_file_path[1:]
                try:
                    ret[relative_file_path] = File(root + os.sep + file)
                except KeyError:
```

```
                # This means the type of file cannot be instantiated into a .json or .h5
                pass
            else:
                pass
    return ret

def _init_all_files_in_database(self):
    self._all_files_in_database = self.explore_folder_recursive(self.database_path)

def add_file(self, relative_file_path, overwrites_file=False):
    """
    :param overwrites_file:
    :param relative_file_path: relative path to file from database
    :return: is_successful
    """
    absolute_file_path = os.path.join(self.database_path, relative_file_path)

    try:
        directory_path = os.path.split(absolute_file_path)[0]
        os.makedirs(directory_path)
    except FileExistsError:
        # The experiment folder already exists, no need to create again
        pass
    else:
        pass

    try:
        self._all_files_in_database[relative_file_path]
    except KeyError:
        file_exists = False
    else:
        file_exists = True
    if file_exists and not overwrites_file:
        raise FileExistsError(f"This file ({relative_file_path}) already exists in the Database folder, maybe set overwrites
        when calling this function")
    elif file_exists and overwrites_file:
        self._all_files_in_database[relative_file_path].delete()

    self._all_files_in_database.update({relative_file_path: File(absolute_file_path)})

def get_file(self, relative_file_path) -> File:
    return self._all_files_in_database.get(relative_file_path)

@property
def used_memory(self):
    return 0

@property
def max_memory(self):
    return Singleton['MAX_FILESYSTEM_MEMORY']

@property
def database_path(self):
    return Singleton['DATABASE_PATH']

def __test__(self):
    self.add_file(f'Hello{os.sep}world{os.sep}this{os.sep}is{os.sep}a{os.sep}test', 'evaluator.h5')
    numpy_file = self.get_file('evaluator.h5')
    # f.__test__()
    print("Testing __iadd__")
    numpy_file += {
        'subset_len_avg': np.zeros(1000000),
        'subset_len': np.ones(1000000),
        'std_dev': np.ones(1000000) * 2,
        'avg_std_dev': np.ones(1000000) * 3
    }
    numpy_file.print_read_file()
    print("subset_len", numpy_file['subset_len'], numpy_file['subset_len'].shape, numpy_file['subset_len'].dtype)

def __getitem__(self, relative_file_path) -> File:
    if self.get_file(relative_file_path=relative_file_path) is None:
        self.add_file(relative_file_path=relative_file_path, overwrites_file=False)
    return self.get_file(relative_file_path=relative_file_path)

@property
def source_json(self) -> File:
```

```
        ret = self[Singleton['CURR_CONFIG_FILE']]
        return ret

    @property
    def selection_json(self) -> File:
        ret = self._all_files_in_database[Singleton['CURR_SELECTION_FILE']]
        return ret

    @property
    def experiment_eater(self) -> None:
        abs_folder_path = Singleton['DATABASE_PATH'] + os.sep + Singleton['EXECUTION_SELECTION_EATER_FOLDER']
        if not os.path.isdir(abs_folder_path):
            os.makedirs(abs_folder_path)
        ret = self.explore_folder_recursive(absolute_path=abs_folder_path)
        if len(ret) == 0:
            raise FileNotFoundError(f"No file could be eaten, please put a valid selection at -> {abs_folder_path}")
        elif len(ret) > 1:
            raise FileNotFoundError(f"Too many files on the eater folder, I need exactly 1 file at -> {abs_folder_path}")
        elif len(ret) == 1:
            source = Singleton['DATABASE_PATH'] + os.sep + list(ret.keys())[0]
            dest = Singleton['DATABASE_PATH'] + os.sep + Singleton['USED_SELECTIONS_EXEC']
            if not os.path.exists(dest):
                os.makedirs(dest)
            file_name = os.path.split(source)[1]
            dest += os.sep + file_name

            if not os.path.exists(dest):
                shutil.copy(source, dest)
            else:
                print(f"WARNING: Please select another name, file already exists at -> {dest}")

            dest = Singleton['DATABASE_PATH'] + os.sep + Singleton['CURR_SELECTION_FILE']
            shutil.move(source, dest)
            print(f"File {file_name} selected succesfully")



        else:
            raise NotImplementedError
        return

# __Filesystem().__test__()
```

20.1.25 Model/Filesystem/File/file.py

```
from abc import ABC, abstractmethod
import os


class File(ABC):
    def __new__(cls, abs_path):
        from Model.Filesystem.File import FILE_CLASSES
        try:
            file_class = FILE_CLASSES[os.path.splitext(abs_path)[1]]
            return super(cls, file_class).__new__(file_class)
        except KeyError:
            raise KeyError

    def __init__(self, abs_path):
        self._abs_path = abs_path
        self._extension = None
        self._content = None

    @abstractmethod
    def size(self):
        pass

    @abstractmethod
    def save(self, *args, **kwargs):
        pass

    @abstractmethod
    def __iadd__(self, other):
        pass

    @staticmethod
```

```
    def load():
        pass

    @property
    def extension(self):
        return self._extension

    def delete(self):
        if os.path.isfile(self._abs_path):
            print("DELETING FILE FROM DATABASE: ", self._abs_path)
            try:
                os.remove(self._abs_path)
            except IsADirectoryError:
                raise IsADirectoryError(f"This error probably means there is a folder with an " +
                                        f"instantiated file extension. ")
```

20.1.26 Model/Filesystem/File/jsonFile.py

```
import json

from Model.Filesystem.File.file import File


class JsonFile(File):
    def __init__(self, path):
        super().__init__(path)
        self._extension = '.json'
        self._content = {}

    def size(self):
        return 100

    def load(self) -> dict:
        try:
            with open(self._abs_path, 'r', encoding='utf8') as file:
                self._content = json.load(file)
        except FileNotFoundError:
            pass
        return self._content

    @staticmethod
    def __merge_second_to_first(a: dict, b: dict, path=None):
        """merges b into a"""
        pop_list = []
        for key, value in b.items():
            if isinstance(key, int):
                pop_list.append((key, value))
        for key, value in pop_list:
            b.pop(key)
            b.update({str(key): value})
            print(f'WARNING: dict contains integer {key}:{value}')

        if path is None:
            path = []

        for key in b:
            if key in a:
                if isinstance(a[key], dict) and isinstance(b[key], dict):
                    JsonFile.__merge_second_to_first(a[key], b[key], path + [str(key)])
                elif a[key] == b[key]:
                    pass  # same leaf value
                else:
                    raise Exception('Conflict at %s' % '.'.join(path + [str(key)]))
            else:
                a[key] = b[key]

        return a

    def __iadd__(self, other: dict) -> 'JsonFile':
        self._content = JsonFile.__merge_second_to_first(self._content, other)
        return self

    def save(self, python_dict):
        with open(self._abs_path, 'w', encoding='utf8') as file:
            json.dump(
                python_dict,
```

```
                file,
                indent=4,
                allow_nan=True,
                sort_keys=True,
                default=lambda o: o.get_json_serializable
            )
```

20.1.27 Model/Filesystem/File/numpyFile.py

```
from abc import ABC

import h5py
import numpy as np
import os

from Model.Filesystem.File.file import File


class NumpyFile(File, ABC):
    def __init__(self, abs_path):
        super().__init__(abs_path)
        self._abs_path = abs_path
        self._file_content = None

    def save(self, python_dict):
        self.__iadd__(python_dict)

    def size(self) -> int:
        if os.path.isfile(self._abs_path):
            return os.path.getsize(self._abs_path)
        else:
            return 0

    def __iadd__(self, python_dict: {str: np.ndarray}) -> 'NumpyFile':
        """
        Transforms a dict representing a subfolder structure with np.ndarrays as nodes into a file
        """
        return self.__iadd_recursive(python_dict=python_dict)

    def __iadd_recursive(self, python_dict: dict, path=f""):
        initial_size = self.size()
        print('++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++')
        for idx, (key, item) in enumerate(python_dict.items()):
            current_path = f'{path}{os.sep}{self.__dict_key2str(key)}'
            if isinstance(item, dict):
                self.__iadd_recursive(item, path=current_path)
            elif isinstance(item, np.ndarray):
                with h5py.File(self._abs_path, 'a') as f:
                    if key not in f.keys():
                        dset = f.create_dataset(path+os.sep+str(key), data=item, shape=item.shape, dtype=item.dtype, maxshape=(
                                                chunks=True, compression="gzip")
                    else:
                        print("KEY: ", key)
                        dset = f[key]
                        if item.dtype == dset.dtype:
                            initial_shape = dset.shape[0]
                            f[key].resize((dset.shape[0] + item.shape[0],))
                            dset[initial_shape:initial_shape + item.shape[0]] = item[:]
                        else:
                            raise TypeError
            else:
                raise Exception

        print(f"Size increment: {self.size() - initial_size}")
        print('------------------------------------------------------------------------')
        return self

    @staticmethod
    def __dict_key2str(key) -> str:
        if isinstance(key, str):
            return key
        else:
            print('WARNING: the key of the folder wasn\'t a string')
            raise Exception
            return str(key)
```

```
    def print_read_file(self):
        with h5py.File(self._abs_path, 'r') as f:
            for key in f.keys():
                print(f"{key} ---- {np.array(f[key])}")
                print("size", len(f[key]))

    def keys(self):
        ret = None
        ret = self._visit_all()
        return ret

    def __getitem__(self, key):
        with h5py.File(self._abs_path, 'r') as f:
            return np.array(f[key])

    def _visit_all(self):
        with h5py.File(self._abs_path, 'r') as f:
            f.visititems(self._visitor_func)
            ret = self._file_content
            self._file_content = None
            return ret

    def _visitor_func(self, name, node):
        if isinstance(node, h5py.Dataset):
            if self._file_content is None:
                self._file_content = []
            self._file_content.append(name)
        else:
            pass
```

20.1.28 Utils/Timer/timer.py

```
import time


def measure_time(func):
    """
    https://pythonbasics.org/decorators/
    """

    def wrapper(*arg):
        t = time.time()
        res = func(*arg)
        # print(f'Function {func.__module__} took  {time.time() - t} seconds to run')
        from Controller.Timer import Timer
        Timer.update({f'{func.__module__}.{func.__name__}': time.time() - t})
        return res
    return wrapper


class __Timer(dict):
    pass


@measure_time
def test():
    print("Hello world")
```

20.1.29 Utils/ValueDomain/valueDomain.py

```
class ValueDomain:
    def __init__(self, *, value_type, bounds: list[callable] = [], default=None):
        self._bounds = bounds
        self._value_type = value_type
        self._default = default
        self._current_value = None

    def __call__(self, value):
        for bound in self._bounds:
            if not bound(value):
                raise ValueError(f'Value {value} is not in bounds')
        return self._value_type(value)  # Try to cast into value type, error not handled

    def __repr__(self):
        return f'ValueDomain(bounds:{self._bounds}, value_type: {self._value_type}, default: {self._default})'
```

```
    def set_current_value(self, value):
        self._current_value = self(value)

    @property
    def get_json_serializable(self):
        return self._default

    @property
    def default(self):
        if self._current_value is None:
            return self._default
        return self._current_value
```

20.1.30 Utils/check_all_supers.py

```
# from Controller.Algorithm.affirmativeSamplingAlgorithm import AffirmativeSampling
# from Controller.Algorithm.affirmativeSamplingWindowedAlgorithm import AffirmativeSamplingWindowed

#
# a = AffirmativeSamplingWindowed
import sys

VERBOSE = False


def check_has_serializable_super_recursive(cls) -> []:
    res = []
    if VERBOSE:
        print("bases", cls.__bases__, ':', type(cls.__bases__))
    for base in cls.__bases__:
        if VERBOSE:
            print("base", base)
        if VERBOSE:
            print("base dict", base.__dict__)
        res += [base]
        res += check_has_serializable_super_recursive(base)
    return res


def merge_dicts(dict_one_aux, dict_two):
    dict_one = {key: item for key, item in dict_one_aux.items()}
    for key, value in dict_two.items():
        try:
            dict_one[key] = value
        except Exception:
            raise NotImplementedError
    return dict_one


def serialize_supers(cls, path=None, depth=4) -> dict:
    from Utils.serializableClass import SerializableClass
    if path is None:
        path = []

    if depth == 0:
        raise RecursionError('Exploration too deep')

    while not (SerializableClass in cls.__bases__):
        for base in cls.__bases__:
            if VERBOSE: print("base", base)
            new_path = [item for item in path]
            new_path += [base]
            if VERBOSE: print("newpath", new_path)
            return serialize_supers(base, path=new_path, depth=depth - 1)

    if issubclass(cls, SerializableClass):
        if VERBOSE: print("FINAL PATH", path)
        res = {}
        for item in path[::-1]:
            res = merge_dicts(res, item.serialized_class())

        return res


def search_module(dictionary, name):
    for key, item in dictionary.items():
```

```
        if VERBOSE: print(f'{key} : {type(key)} <-> {item} : {type(item)}')
        if VERBOSE: print("SPLIT:", key.split('.')[-1][1:], '==', name[1:])
        if key.split('.')[-1][1:] == name[1:]:
            if VERBOSE: print("FOUND!!!!")
            return item


def compare_merge(original: dict, diff: dict) -> dict:
    """
    :param original: original dictionary (contains all keys)
    :param diff: modifications over original dict (key - value)
    :return: The new dictionary with the diff changes committed into the original.
    """
    ret = {key: item for key, item in original.items()}
    original_keys = original.keys()
    for key, value in diff.items():
        if key in original_keys:
            ret[key] = value
        else:
            raise KeyError

# print("Check recursive", check_has_serializable_super_recursive(a))
# print("serialize_supers", serialize_supers(a))
# print(merge_dicts({}, {1:a}))
# search_module('AffirmativeSampling')
```

20.1.31 Utils/f_distribution.py

```
import numpy as np
from scipy.stats import f


def f_test(x: np.ndarray, y: np.ndarray) -> (float, float):
    """
    :param x: variance of random variable
    :param y: variance of random variable
    :return:
    (
        f_ratio: the F ratio >= 0 (because variance is positive), the closer F ratio is to 1, the better,
        p_value: the confidence level, the closer is to 1 the better, it symbolizes the conficence level
    """
    x = np.array(x)
    y = np.array(y)
    f_test_statistic = np.var(x)/np.var(y)
    degrees_freedom_numerator = x.size-1
    degrees_freedom_denominator = y.size-1
    assert degrees_freedom_numerator >= degrees_freedom_denominator
    p_value_of_f_test_statistic = 1-f.cdf(f_test_statistic, degrees_freedom_numerator, degrees_freedom_denominator)
    return f_test_statistic, p_value_of_f_test_statistic
```

20.1.32 Utils/lcm.py

```
from functools import reduce
from math import gcd


def lcm(numbers):
    return reduce((lambda x, y: int(x * y / gcd(x, y))), numbers)
```

20.1.33 Utils/path_in_dict.py

```
import re


def path_in_dict(dict_object, path):
    keys = re.split(r'/', path)[1:]
    try:
        curr_dict = dict_object[keys[0]]
        for key in keys[1:]:
            curr_dict = curr_dict[key]
    except KeyError:
        cpp = check_possible_paths(dict_object, keys[-1], '')
        if len(cpp) >= 1:
            raise ValueError(f"The path doesn't exist, did you mean {cpp}")
        else:
```

```
            raise ValueError(f'Path doesn\'t exist, no suggestions')
    return True


def check_possible_paths(dict_object, name, path):
    ret = []
    if isinstance(dict_object, dict):
        for key in dict_object.keys():
            if key == name:
                ret.append(path + '/' + key)
            else:
                ret += check_possible_paths(dict_object[key], name, path + '/' + key)
    return ret
```

20.1.34 Utils/serializableBuilderCatalog.py

```
import Utils.serializableClass

BUILDER_TYPES = {'unique', 'multiple'}


class SerializableBuilderCatalog:
    def __init__(self, *classes, builder_type='unique'):
        self._serialized_dict = {}
        self._subclasses = {}

        for cls in classes:
            self._subclasses[cls.__name__] = cls
            if issubclass(cls, Utils.serializableClass.SerializableClass):
                self._serialized_dict[cls.__name__] = cls.serialized_class()
            else:
                raise ValueError
        if builder_type in BUILDER_TYPES:
            self._serialized_dict['_builder_type'] = builder_type
            self._builder_type = builder_type
            if self._builder_type == 'unique':
                self._instance_content = None  # TODO: this will store instances when setting attribute
            elif self._builder_type == 'multiple':
                self._instance_content = []

        else:
            raise ValueError

    def __repr__(self):
        return str(self._serialized_dict)

    @property
    def serialized_dict(self) -> dict:
        return self._serialized_dict

    def __getitem__(self, key: str):
        """
        :param key: name of the class
        :return: generic class that can be initialized into instances.
        """
        return self._serialized_dict[key]

    def __set_name__(self, owner, name):
        """
        Magic method for descriptors >python3.6 to increase naming coherence
        :param owner: This is a class, not the instance of the class
        :param name: name of the attribute we are declaring on a class level
        :return:
        """
        self.name = name

    def __get__(self, instance, owner):
        if instance is None:
            return self
        return instance.__dict__[self.name]

    def __set__(self, instance, value) -> None:
        if self._builder_type == 'unique':
            for cls in self._subclasses.values():
                if isinstance(value, cls):
                    instance.__dict__[self.name] = value
```

```
                    return
                raise ValueError(f'The class {type(value)} you are trying to set doesn\'t belong to builder catalog')
            elif self._builder_type == 'multiple':
                if isinstance(value, list):
                    for cls in self._subclasses.values():
                        for idx, val in enumerate(value):
                            if isinstance(val, cls):
                                pass
                            elif idx == len(self._subclasses) - 1:
                                raise ValueError(f'The class {type(val)} you are trying to set doesn\'t belong to builder catalog')

                    instance.__dict__[self.name] = value
                else:
                    raise ValueError(f'Expecting an array of objects, got {list} {type(value)}')
            else:
                raise Exception('Builder type doesn\'t exist')



        @property
        def subclasses(self):
            return self._subclasses
```

20.1.35 Utils/serializableClass.py

```
from Utils.serializableClassAttribute import SerializableClassAttribute
from Utils.serializableBuilderCatalog import SerializableBuilderCatalog


class SerializableClass:
    def serialized_instance(self) -> dict:
        return {
            key: self.__dict__[key]
            for key in self.__dict__['_serialize']
        }

    @classmethod
    def serialized_class(cls) -> dict:
        ret = {}
        for key, value in cls.__dict__.items():
            if issubclass(type(value), SerializableClassAttribute):
                if not value.is_serializable_class:
                    if value.is_value:
                        ret[key] = value.default
                    else:
                        ret[key] = None

                if value.is_serializable_class:
                    ret[key] = value.serialized_class()

            elif issubclass(type(value), SerializableBuilderCatalog):
                for key1, value1 in value.serialized_dict.items():
                    if isinstance(value1, dict):
                        try:
                            ret[key][key1] = "This means I didn't overwrite this value"
                        except KeyError:
                            ret[key] = {}
                        finally:
                            if issubclass(type(value1), SerializableClassAttribute):
                                ret[key][key1] = value1.serialized_class()
                            else:
                                ret[key][key1] = value1

                    else:
                        ret[key][key1] = value1
        return ret
```

20.1.36 Utils/serializableClassAttribute.py

```
from Utils.ValueDomain.valueDomain import ValueDomain


class SerializableClassAttribute:
    """
    This class will be used as a Descriptor for attributes.
    Classes that implement this class on an instance level will inform
```

```
the Config work flow to store class information in a JSON format
which will be human-readable. This way to store experiment DATA in a
transparent format will help document all experiments in a consistent
way in order for the experiments to be repeatable. This transparency will
give our conclusions a solid background.

"""

def __init__(self, *, name=None, is_serializable_class=None, is_value: ValueDomain = None):
    """
    :param name: Not necessary because of magic method set_name
    :param is_serializable_class: reference to class NODE instead of VALUE node
    :param is_value: in case is_serializable_class is None, instance of type ValueDomain
    """
    self.name = name
    self._is_serializable_class = is_serializable_class
    self._is_value = is_value

def __set_name__(self, owner, name):
    """
    :param owner: This is a class, not the instance of the class
    :param name: name of the attribute we are declaring on a class level
    :return:
    """
    self.name = name

def __get__(self, instance, owner):
    """
    Required in PEP to implement a data descriptor (serialize data)
    """
    if instance is None:
        return self

    ret = None
    try:
        ret = instance.__dict__[self.name]
    except KeyError:
        raise KeyError
    return ret

def __set__(self, instance, value) -> None:
    # In case it's a value belonging to ValueDomain I have to check value is within domain
    if self.is_value:
        self._is_value.set_current_value(value)

    # I put the value in the dictionary of the object instance
    instance.__dict__[self.name] = value

    # I put the name in the _serialize array to inform serializer it has to serialize it
    try:
        instance.__dict__['_serialize']
    except KeyError:
        instance.__dict__['_serialize'] = []
    instance.__dict__['_serialize'].append(self.name)
    return None

@property
def is_serializable_class(self):
    return self._is_serializable_class is not None

def serialized_class(self) -> dict:
    from Utils.serializableClass import SerializableClass
    if self.is_serializable_class:
        if issubclass(self._is_serializable_class, SerializableClass):
            return self._is_serializable_class.serialized_class()
        else:
            raise Exception

def __repr__(self):
    return f'SerializableClassAttribute( is_serializable_class={self._is_serializable_class}, is_value:{self._is_value}) '

@property
def is_value(self):
    return self._is_value is not None

@property
```

```
    def value(self):
        return self._is_value

    @property
    def default(self):
        return self.value
```

20.1.37 Utils/string_to_binary.py

```
import binascii


def string_to_binary(item: str) -> bytes:
    return binascii.a2b_qp(item)
```

20.1.38 View/DatabaseDrawer/databaseDrawer.py

```
import os
import re
import numpy as np
from itertools import product

from Model.Filesystem.File.numpyFile import NumpyFile
from View.Plotting.scatter import Scatter
from Controller.Singleton import Singleton
from Model.Filesystem import Filesystem
from Utils.f_distribution import f_test
from View.Plotting.histogram import Histogram


class DatabaseDrawer:
    def __init__(self):
        self._properties = [
            'subset size',
            'intersection size',
            'avg subset size',
            'avg intersection size',
            'f distribution subset size'
        ]

    def __call__(self):
        self.start(Singleton['DATABASE_PATH'])

    def start(self, path):
        for file in os.listdir(path):
            current_path = os.path.join(path, file)
            if os.path.isdir(current_path) and file[0] != '.':
                print(f"Directories: {file}")
                self.start(current_path)
            elif os.path.isfile(current_path):
                try:
                    file = Filesystem[current_path]
                except KeyError:  # file doesn't exist
                    pass
                except NotImplementedError:  # format is not implemented
                    pass
                else:
                    if isinstance(file, NumpyFile):
                        print("DRAW NUMPY FILE:  ", current_path)
                        self.__draw_numpy_file(file, current_path)
                    else:
                        print("WARNING: I shouldn't be inspecting non valid files, only .h5")
                        raise Exception

    def __draw_numpy_file(self, numpy_file, name):
        scatter_plot = Scatter('#words', '#distinct')
        histogram_plot = Histogram(name_x='variance', name_y="density")
        max_len = max([numpy_file[key].shape[0] for key in numpy_file.keys()])
        # print("DRAW MAX LEN: ", max_len)

        for prop_key in self._properties:

            # CASE BASIC PROPERTY
            pattern = re.compile(re.escape(prop_key))
            for key in numpy_file.keys():
                if pattern.search(key):
```

```
                scatter_plot.add_line(key, numpy_file[key], max_len)

        # CASE AVG OF BASE PROPERTY
        pattern = re.compile(r'avg ')
        if pattern.search(prop_key):
            property_name = re.split(pattern, prop_key)[1]
            pattern = re.compile(re.escape(property_name))
            for key in numpy_file.keys():
                if pattern.search(key):
                    h5_database_name = re.split(r'/', key)[1]
                    scatter_plot.add_line(prop_key + '/' + h5_database_name, self._get_mean(numpy_file[key]),
                                          max_len, line_width=1.2, linestyle='dashdot', is_avg=True)

        # CASE F DISTRIBUTION OF BASE PROPERTY
        pattern = re.compile(r'f distribution ')
        if pattern.search(prop_key):
            property_name = re.split(pattern, prop_key)[1]
            pattern = re.compile(re.escape(property_name))
            pair_name_array = []
            for key in numpy_file.keys():
                if pattern.search(key):
                    h5_database_name = re.split(r'/', key)[1]
                    pair_name_array.append((h5_database_name, numpy_file[key]))
                else:
                    pass
            all_pair_combinations = list(product(pair_name_array, pair_name_array))
            triangle_no_diag_pair_combinations = [[(name_1, file_1), (name_2, file_2)]
                                                  for (name_1, file_1), (name_2, file_2) in all_pair_combinations
                                                  if name_1 < name_2]
            for tuple_1, tuple_2 in triangle_no_diag_pair_combinations:
                name_1, file_1 = tuple_1
                name_2, file_2 = tuple_2
                histogram_plot.add_line(key=property_name + "/" + name_1, array=self._get_cumulative_var(file_1))
                histogram_plot.add_line(key=property_name + "/" + name_2, array=self._get_cumulative_var(file_2))
                f_test_statistic, p_value = self._get_f_distribution(file_1, file_2)
                if self._check_valid_p_value(p_value=p_value):
                    histogram_plot.set_passes_test()
                histogram_plot.set_f_test_statistic(f_test_statistic=f_test_statistic)
                histogram_plot.set_p_value(p_value=p_value)
            histogram_plot.plot(name + "_hist")
    scatter_plot.plot(name)

def _get_mean(self, numpy_ndarray):
    ret = np.ones_like(numpy_ndarray, dtype=float) * -1
    for idx, item in enumerate(numpy_ndarray):
        if idx > 0:
            ret[idx] = ret[idx - 1] + item
        else:
            ret[idx] = item
    ret = np.divide(ret, np.arange(1, ret.shape[0] + 1))
    return ret

def _get_cumulative_var(self, numpy_ndarray):
    ret = np.ones_like(numpy_ndarray, dtype=float) * -1
    mean = np.mean(numpy_ndarray)
    for idx, item in enumerate(numpy_ndarray):
        ret[idx] = (item - mean) ** 2
    return ret

@staticmethod
def _check_valid_p_value(*, p_value) -> bool:
    if p_value >= Singleton['ALPHA_TEST']:
        return True
    return False

@staticmethod
def _get_f_distribution(x: np.ndarray, y: np.ndarray):
    return f_test(x=x, y=y)

def __test__(self):
    pass

def inspect(self, file):
    pass
```

20.1.39 View/Menu/menu.py

```
from View.Menu.MenuOption.menuOption import MenuOption


class Menu(dict):
    def __init__(self, name):
        super().__init__()
        self._name = str(name)
        self._name2idx = {}
        self._idx2name = {}
        self._current = []
        self._quit = False

    def add_option(self, name, *args, **kwargs):
        idx_menu = len(self) + 1
        self.update({idx_menu: MenuOption(*args, **kwargs)})
        self._name2idx.update({name: idx_menu})
        self._idx2name.update({idx_menu: name})

    def add_submenu(self, name):
        idx_menu = len(self) + 1
        self.update({idx_menu: Menu(name)})
        self._name2idx.update({name: idx_menu})
        self._idx2name.update({idx_menu: name})

    def transition(self, in_value):
        try:
            in_number = int(in_value)
        except ValueError:
            # CASE ITS NOT AN INTEGER (IS A STRING)
            # print("Not int")
            if (in_value == 'Q' or in_value == 'q') and len(self._current) > 0:
                self._current.pop()
            else:
                self._quit = True
        else:
            aux = self
            for idx in self._current:
                aux = aux[idx]

            item = aux.get(in_number)
            if item is not None:
                if isinstance(item, MenuOption):
                    item.action()
                    return
                elif isinstance(item, Menu):
                    self._current.append(in_number)
                    return
            else:
                raise IndexError

    @property
    def quit(self):
        return self._quit

    @property
    def path(self):
        ret = '/'
        aux = self
        for idx, i in enumerate(self._current):
            aux = aux[i]
            ret += f'{aux.name}'
            if idx < len(self._current) - 1:
                ret += '/'
        return ret

    @property
    def name(self):
        return self._name

    def __repr__(self):
        ret = ''
        aux = self
        for idx in self._current:
            aux = aux[idx]
        for i in range(1, len(aux) + 1):
```

```
            ret += f'( {i} ) {(4 - len(str(i))) * "_"} {aux._idx2name[i]}\n'
        return ret

    def __getitem__(self, key):
        ret = None
        if isinstance(key, int):
            item = self.get(key)
            if item is not None:
                ret = item
        else:
            idx = self._name2idx.get(key)
            item = self.get(idx)
            if item is not None:
                ret = item
        return ret
```

20.1.40 View/Menu/MenuOption/menuOption.py

```
class MenuOption:
    """
    This class is a leaf node that performs an action,
    otherwise it's a submenu
    """

    def __init__(self, action=None):
        # self._name = name
        # self._representation = representation
        assert action is not None
        self._action = action

    @property
    def action(self):
        return self._action
```

20.1.41 View/Plotting/histogram.py

```
import matplotlib.pyplot as plt
import numpy as np

import matplotlib as mpl
mpl.rcParams['agg.path.chunksize'] = 10000


class HistogramLine:
    def __init__(self, *, key, axis_x, colors):
        self.key = key
        self.axis_x = axis_x
        self._num_bins = 201
        self._axis_x = self._get_cumulative_dist(array=self.axis_x)
        self._colors = colors
        self._max_value = None
        self.percent_95 = None
        self.percent_1 = None
        self.median = None

    def plot(self, ax1):
        c = next(self._colors)["color"]

        bins = np.linspace(0, self._max_value, self._num_bins)
        sorted_x = np.sort(self.axis_x)

        self.percent_1 = sorted_x[-1]
        self.percent_95 = sorted_x[int(0.95 * len(sorted_x))]
        self.median = sorted_x[int(0.5 * len(sorted_x))]

        ax1.grid()
        ax1.hist(self.axis_x, bins=bins, color=c, alpha=0.3, label=self.key, density=True)
        ax1.axvline(self.median, 0, 1, color=c, linestyle='dashed', label='median', linewidth=0.4)
        ax1.axvline(self.percent_95, 0, 1, color=c, linestyle='dashed', label='95% of items', linewidth=0.7)
        ax1.axvline(self.percent_1, 0, 1, color=c, linestyle='dashed', label='100% of items', linewidth=1.7)

    @staticmethod
    def _get_cumulative_dist(*, array):
        ret = np.zeros_like(array, dtype=np.float)
        ret[:] = array[:]
        aux_sum = 0
```

```
        for item in array:
            aux_sum += item
        ret = np.divide(ret, aux_sum)
        return ret

    def set_max_value(self, max_value):
        self._max_value = max_value


class Histogram(dict):
    def __init__(self, name_x, name_y):
        self.colors = plt.rcParams["axes.prop_cycle"]()
        super(Histogram, self).__init__()
        self._name_x = name_x
        self._name_y = name_y
        self._max_value = -1
        self._background_color = 'lightsalmon'
        self._f_test_statistic = None
        self._p_value = None

    def plot(self, name):
        fig, (ax1) = plt.subplots(1)
        for line in self.values():
            line.set_max_value(self._max_value)
            line.plot(ax1)

        ax1.set_xlabel(self._name_x)
        ax1.set_ylabel(self._name_y)
        plt.legend()
        ax1.set_title(f'F distribution test |F test stat: {"{:.5f}".format(self._f_test_statistic)} | P_value: {self._p_value}')
        plt.savefig(name + '.svg', dpi=100, facecolor=self._background_color)
        plt.close('all')

    def set_f_test_statistic(self, *, f_test_statistic):
        self._f_test_statistic = f_test_statistic

    def set_p_value(self, *, p_value):
        self._p_value = p_value

    def set_passes_test(self):
        self._background_color = 'palegreen'

    def add_line(self,
                 *,
                 key,
                 array):
        if self.get(key):
            raise KeyError(key)
        else:
            self[key] = HistogramLine(key=key, axis_x=array, colors=self.colors)
            self._max_value = max(self._max_value, np.max(array))
```

20.1.42 View/Plotting/scatter.py

```
import random
import time
from collections import defaultdict
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt
import math
import numpy as np
import pandas as pd

import matplotlib as mpl
mpl.rcParams['agg.path.chunksize'] = 10000


class ScatterLine:
    def __init__(self, key, array, size_x, colors, line_width=0.2, linestyle='solid'):
        self.key = key
        self.axis_y = array
        compression = size_x // array.shape[0]
        # print(f"compression {compression}")
        self.axis_x = np.arange(1, size_x + 1, step=compression)
        # print(self.axis_x.shape)
        # print(self.axis_y.shape)
```

```
        self._colors = colors
        self._line_width = line_width
        self._linestyle = linestyle

    def plot(self, ax1):
        # xandy = [(x, y) for x, y in zip(self.axis_x, self.axis_y)]
        # xandy = sorted(xandy, key=lambda item: item[0])
        # axis_x, axis_y = list(zip(*xandy))
        plt.sca(ax1)
        ax1.grid()

        # ax1.plot(axis_x, axis_y, 'bx', label=self.key)
        c = next(self._colors)["color"]
        ax1.plot(self.axis_x, self.axis_y, color=c, label=self.key, linestyle=self._linestyle, linewidth=self._line_width)
        # ax1.scatter(axis_x, axis_y, color=c)  # dots


class Scatter(dict):
    def __init__(self, name_x, name_y):
        self.colors = plt.rcParams["axes.prop_cycle"]()
        self.colors_avg = plt.rcParams["axes.prop_cycle"]()
        super(Scatter, self).__init__()
        self._name_x = name_x
        self._name_y = name_y

    def plot(self, name):
        # fig, (ax1) = plt.subplots(1, figsize=(40, 40))
        fig, (ax1) = plt.subplots(1)
        for line in self.values():
            line.plot(ax1)
        ax1.set_title('Algorithm comparison')
        ax1.set_xlabel(self._name_x)
        ax1.set_ylabel(self._name_y)
        plt.legend()
        plt.savefig(name + '.svg', dpi=1000)
        plt.close('all')

    def add_line(self, key, array, max_len, line_width=0.2, linestyle='solid', is_avg=False):
        aux_colors = None
        if is_avg:
            aux_colors = self.colors_avg
        else:
            aux_colors = self.colors
        if self.get(key):
            raise KeyError(key)
        else:
            self.update({key: ScatterLine(key, array, max_len, aux_colors, line_width=line_width, linestyle=linestyle)})
```

20.2 Experiment figures

20.2.1 Test 1

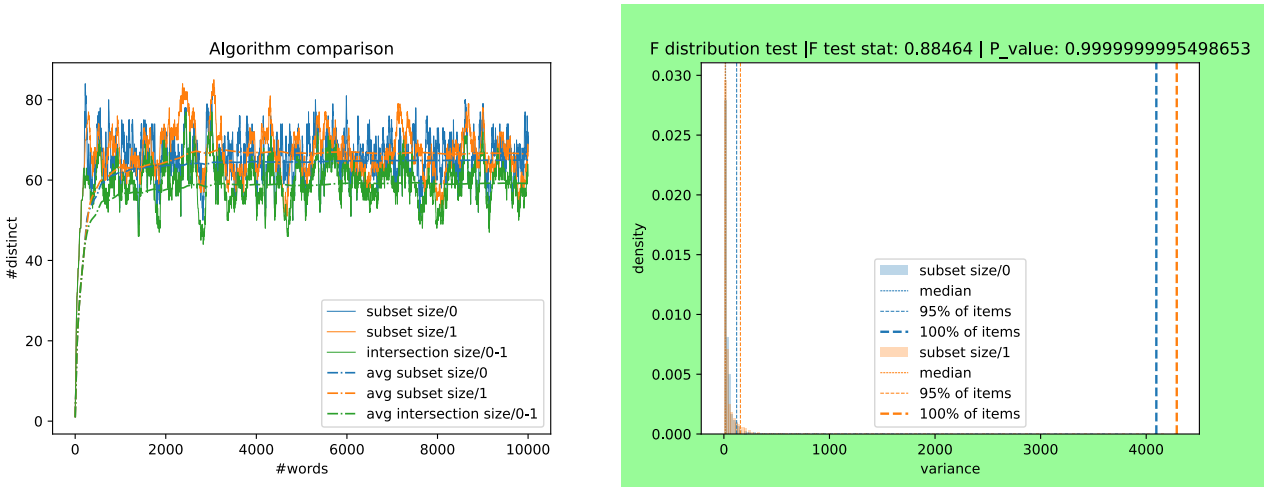


Figure 3: 0.h5

20.2.2 Test 2

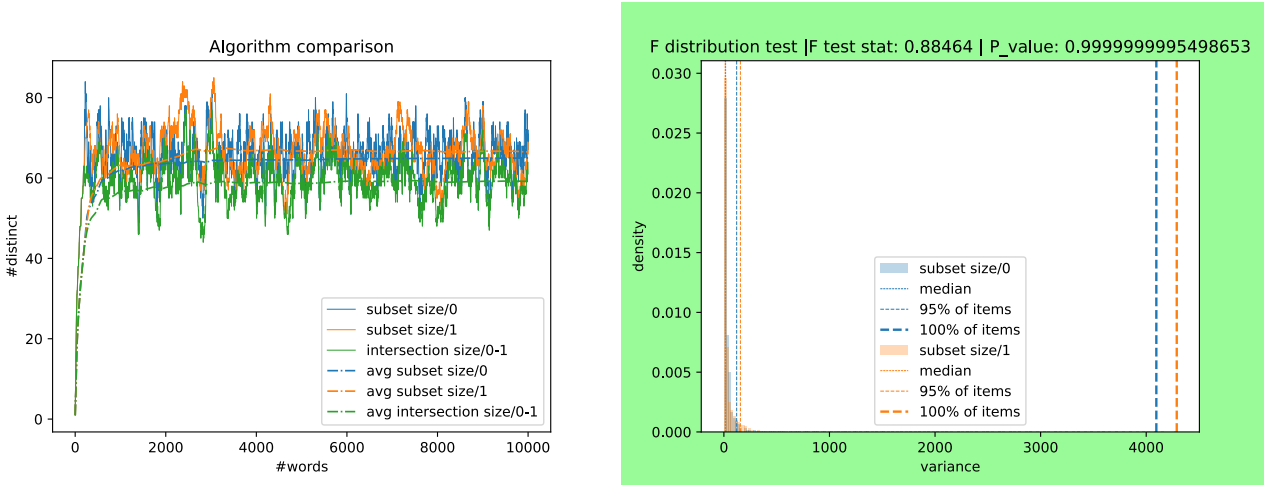


Figure 4: 0_0.h5

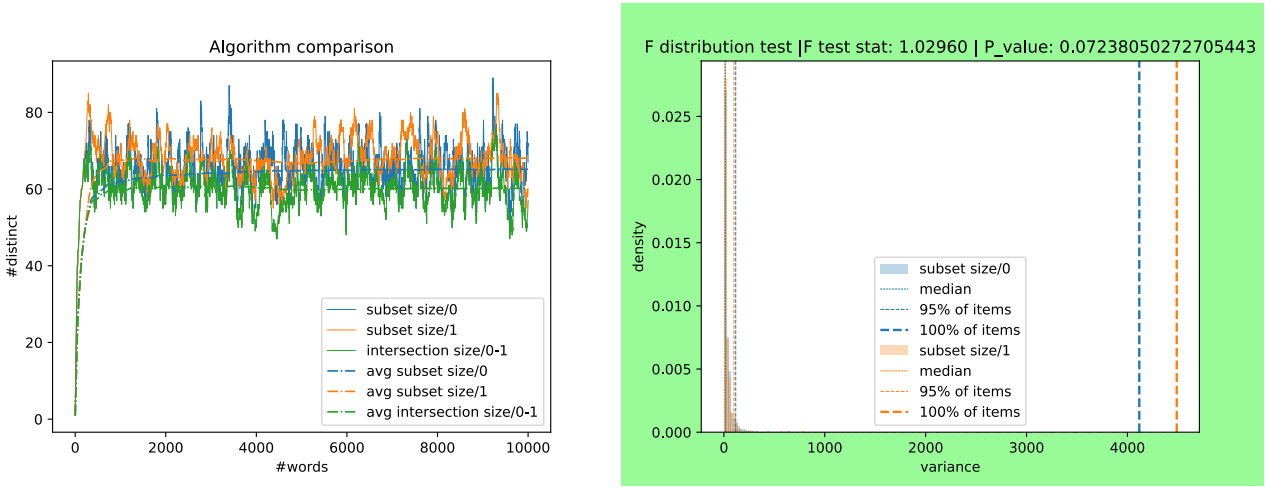


Figure 5: 0_1.h5

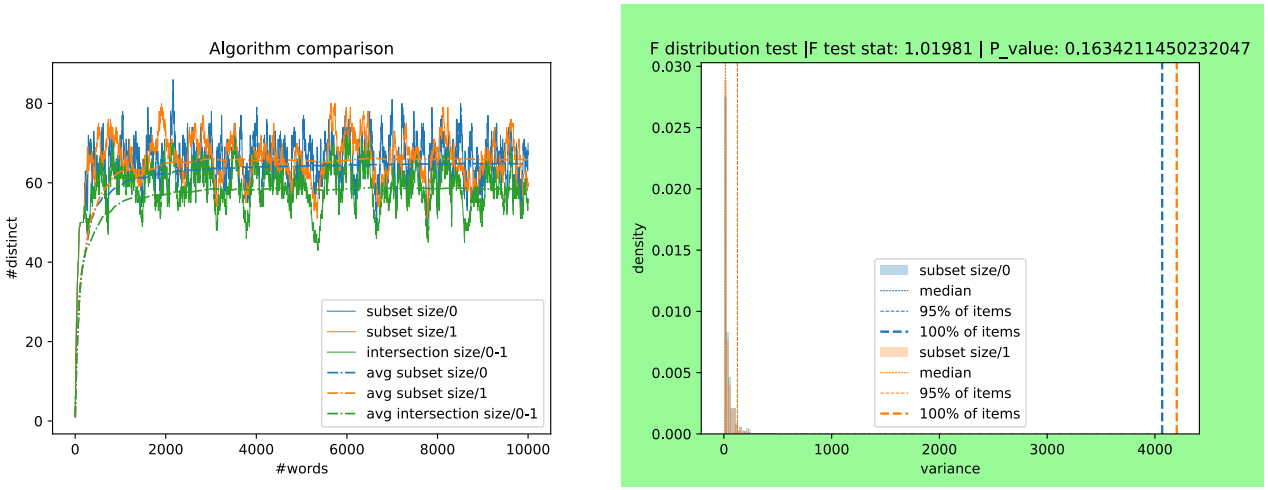


Figure 6: 0_2.h5

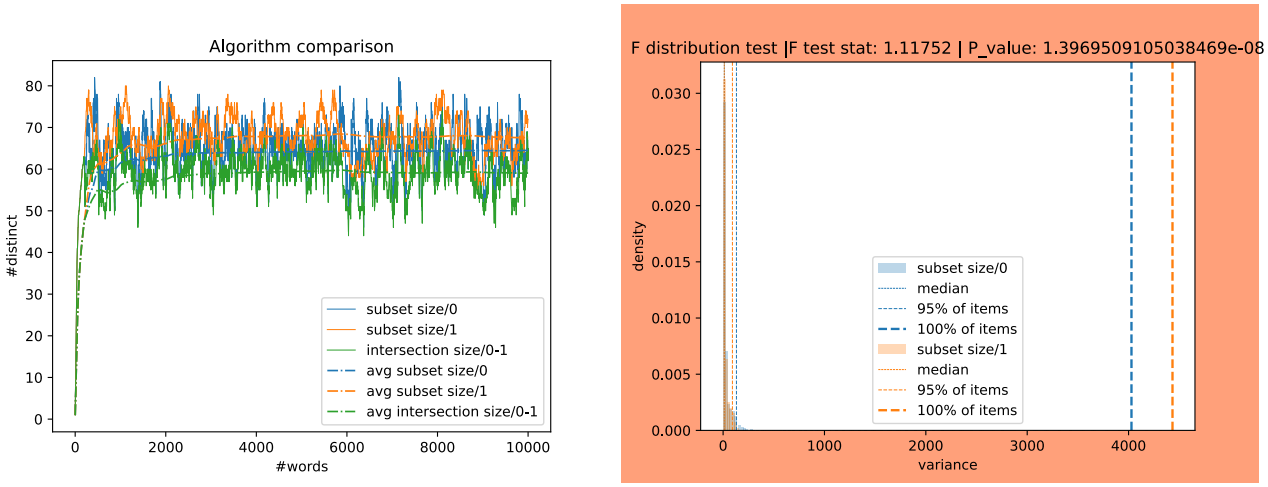


Figure 7: 1_0.h5

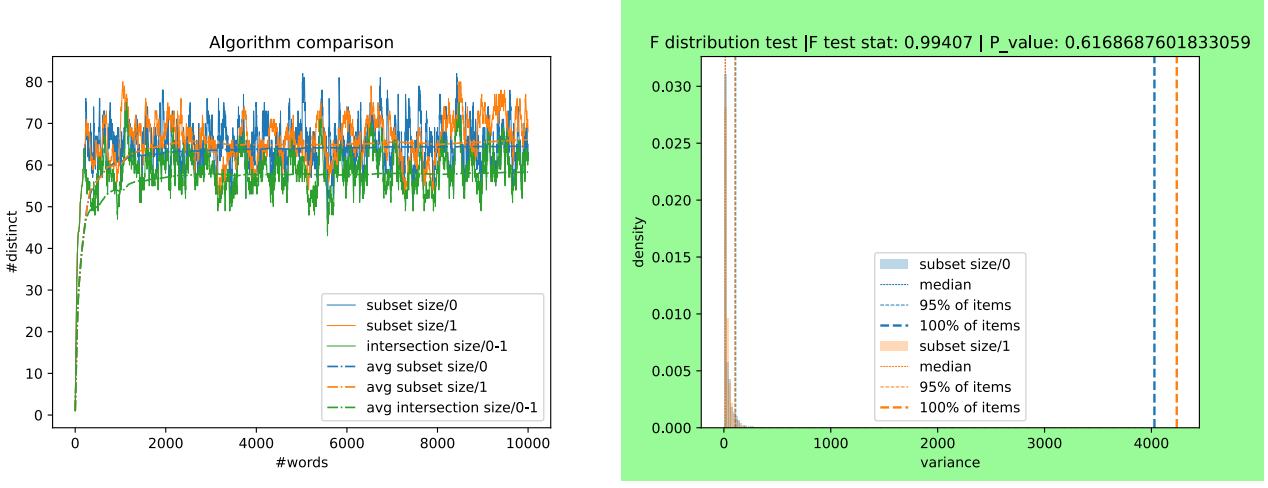


Figure 8: 1_1.h5

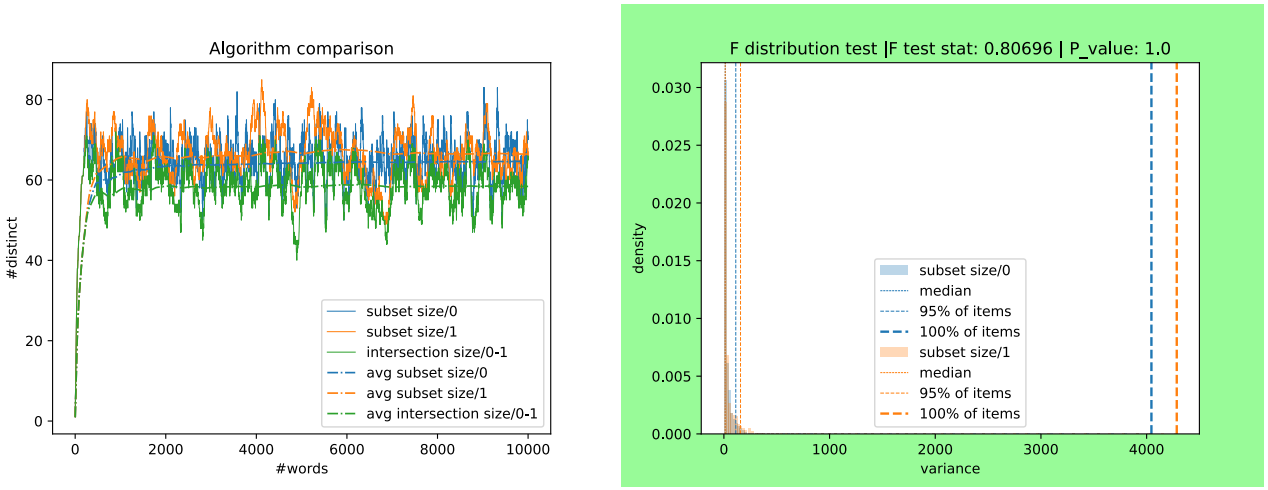


Figure 9: 1_2.h5

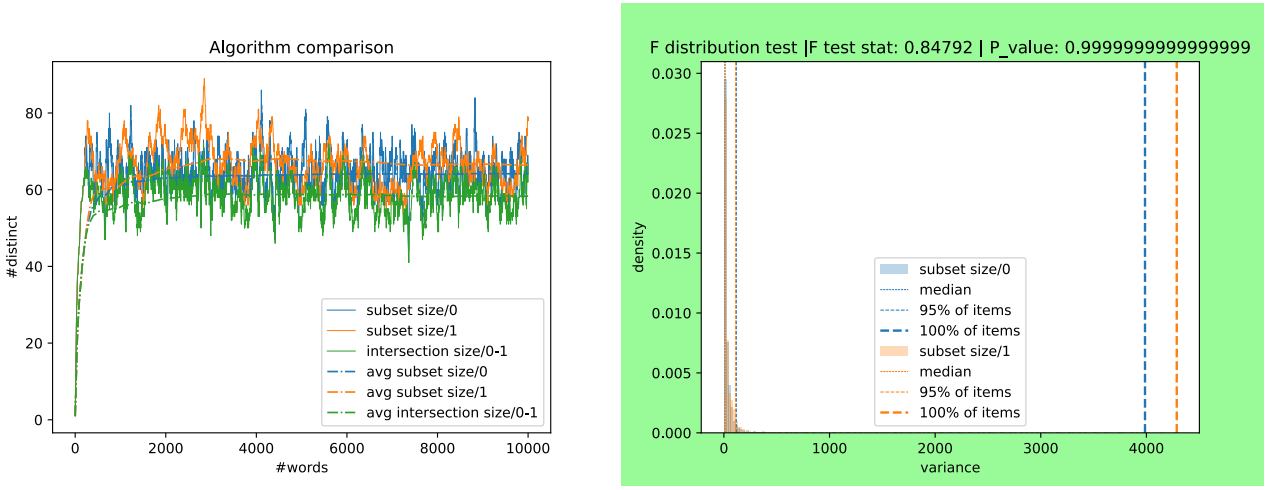


Figure 10: 2_0.h5

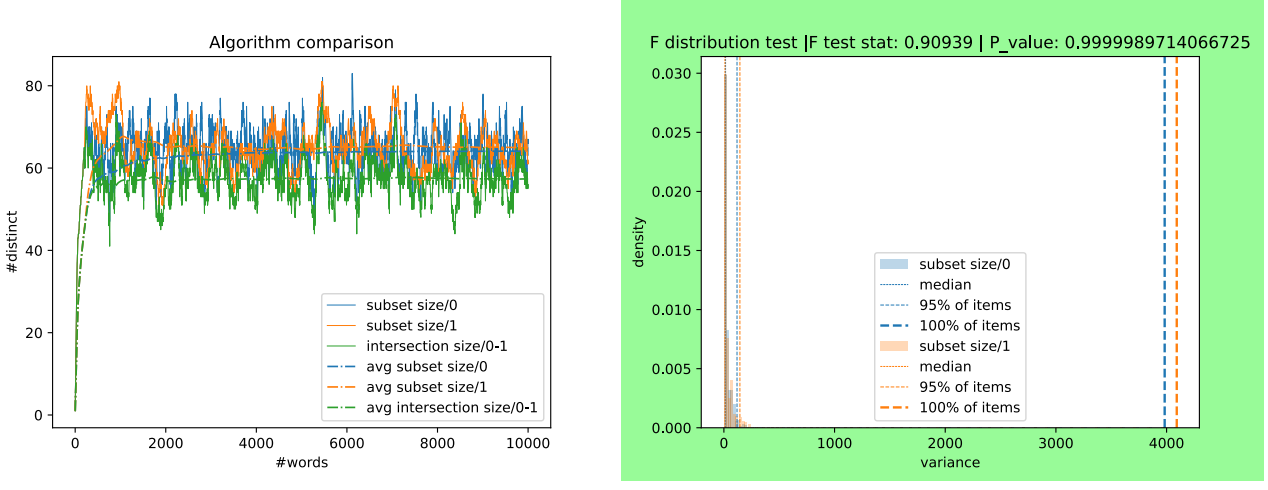


Figure 11: 2_1.h5

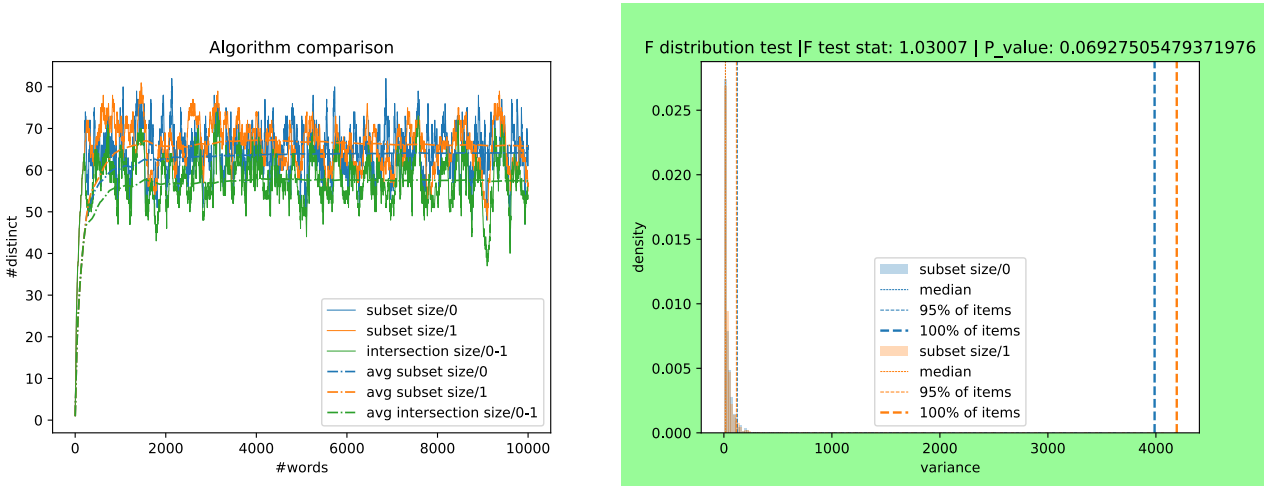


Figure 12: 2_2.h5

20.2.3 Test 3

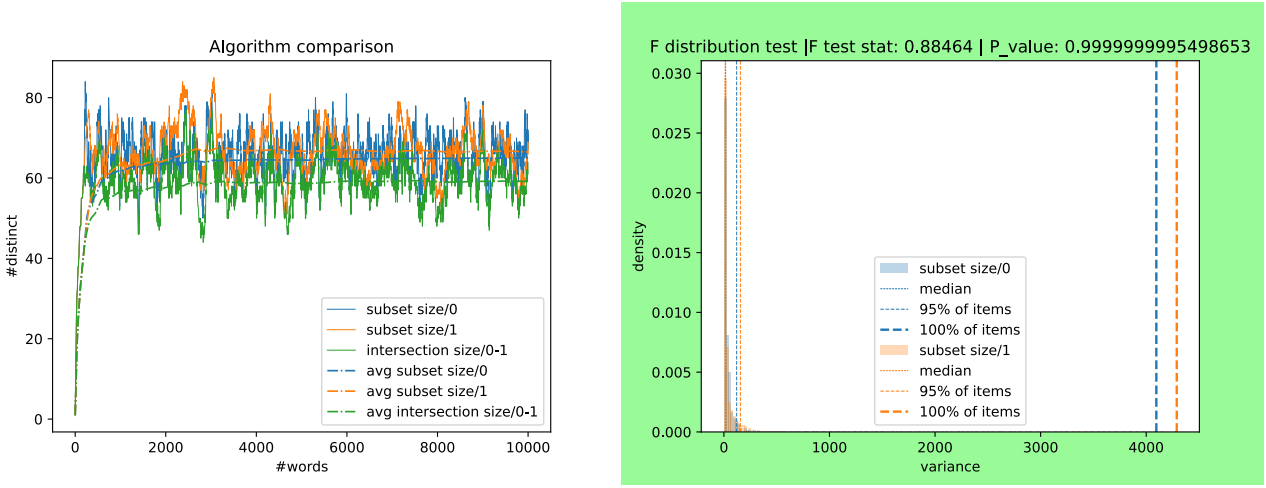


Figure 13: 0_0.h5

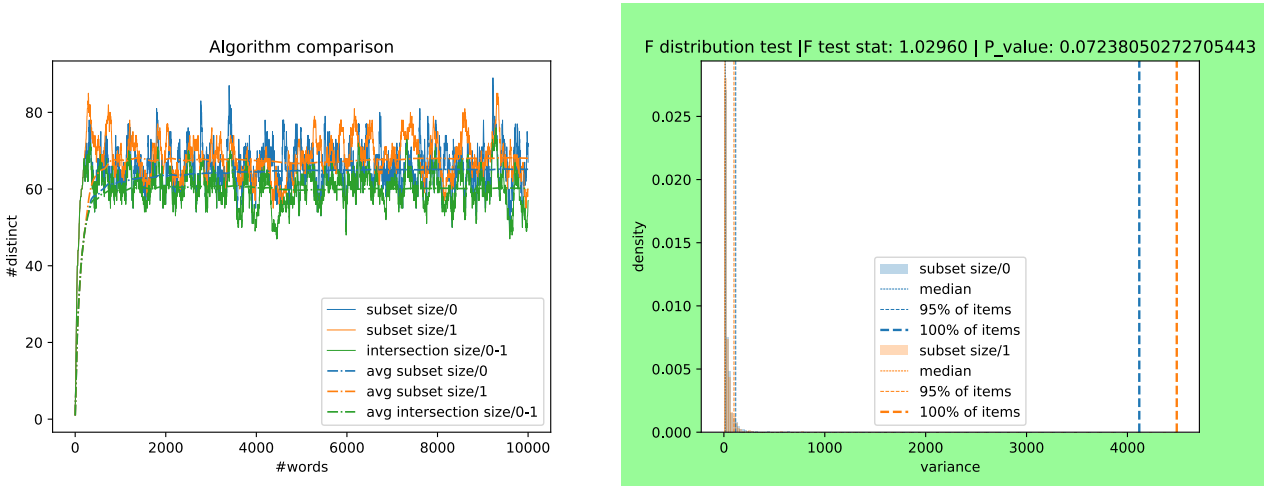


Figure 14: 0_1.h5

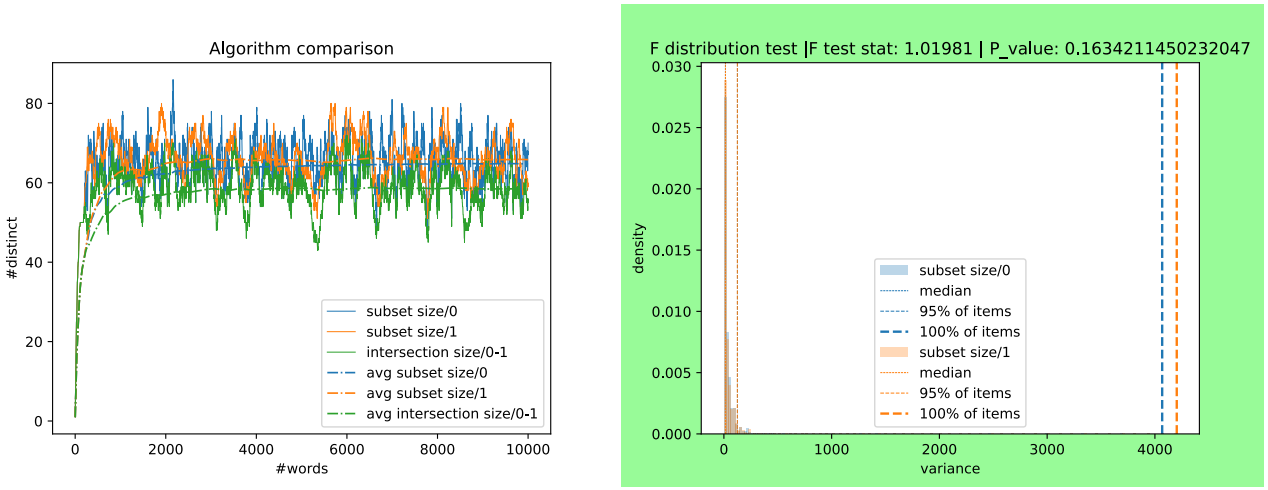


Figure 15: 0_2.h5

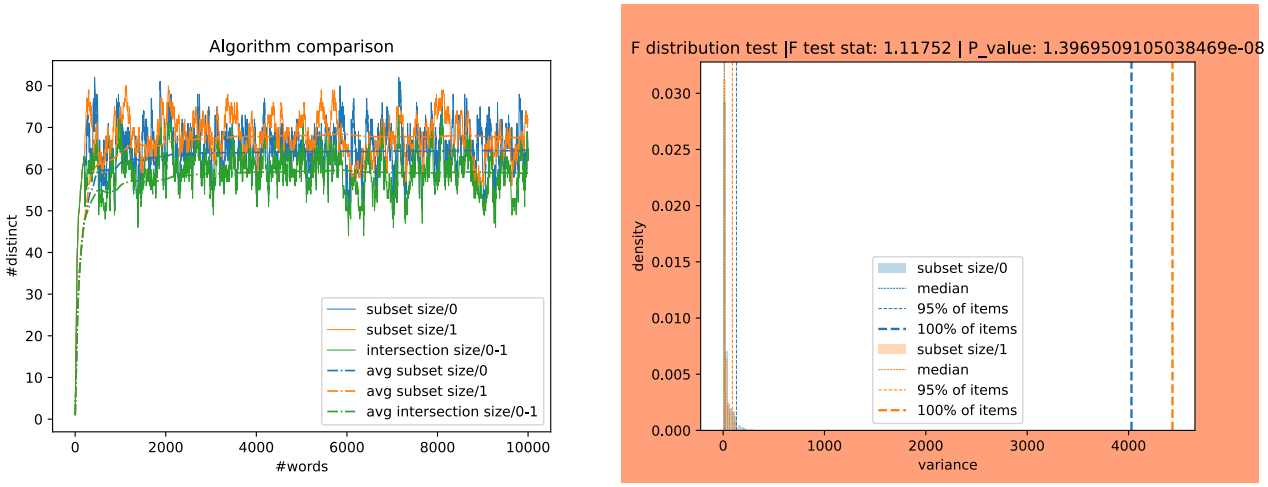


Figure 16: 1_0.h5

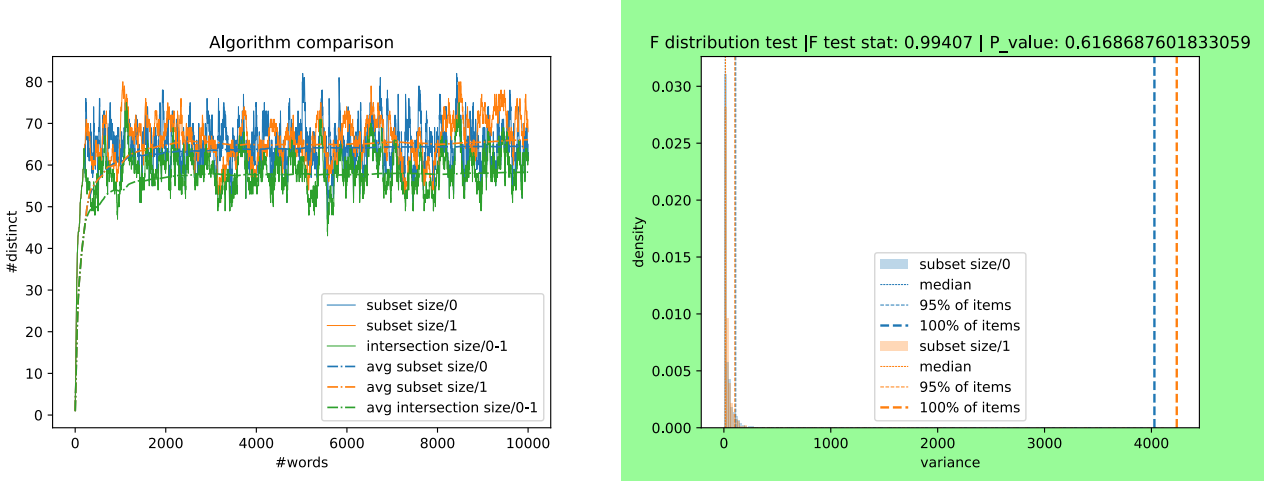


Figure 17: 1_1.h5

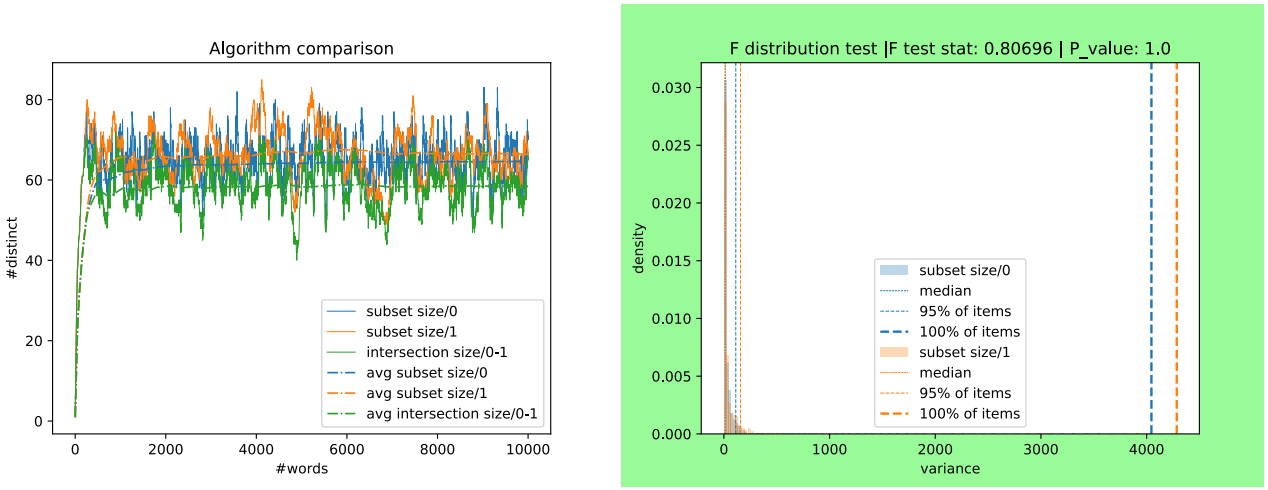


Figure 18: 1_2.h5

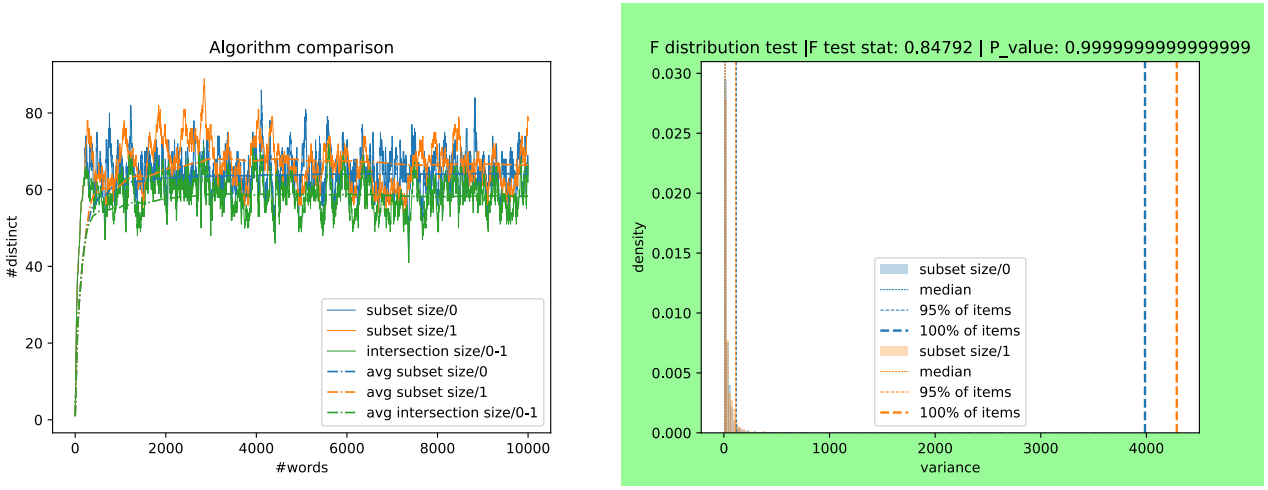


Figure 19: 2_0.h5

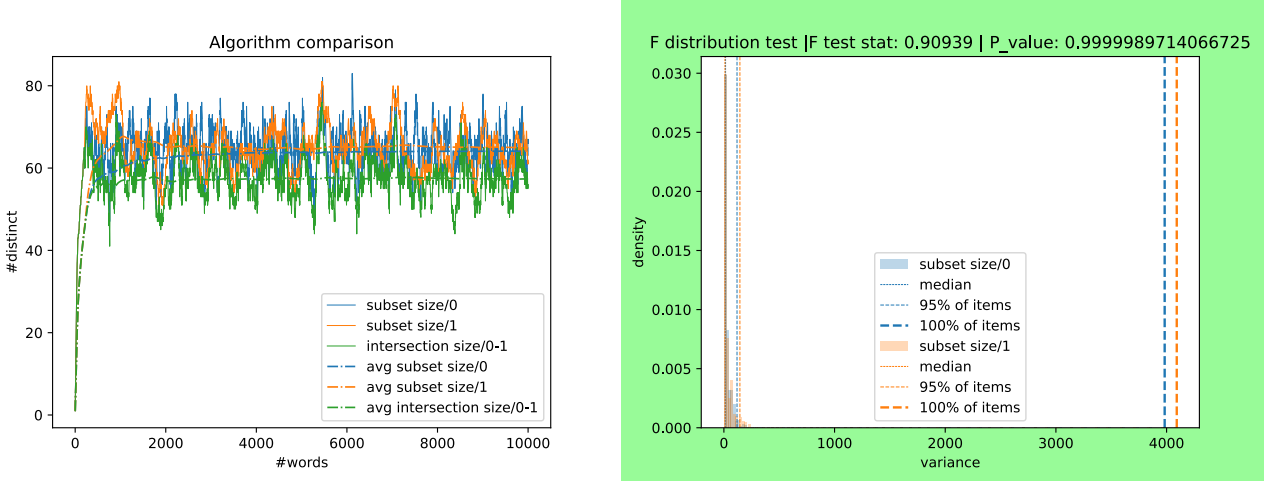


Figure 20: 2_1.h5

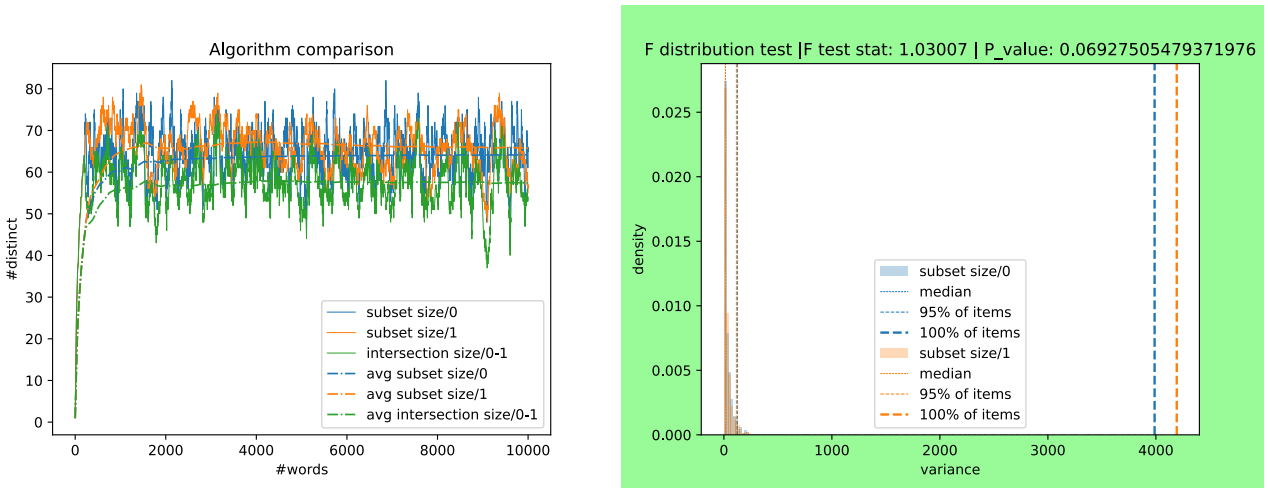


Figure 21: 2_2.h5

20.2.4 Test 4

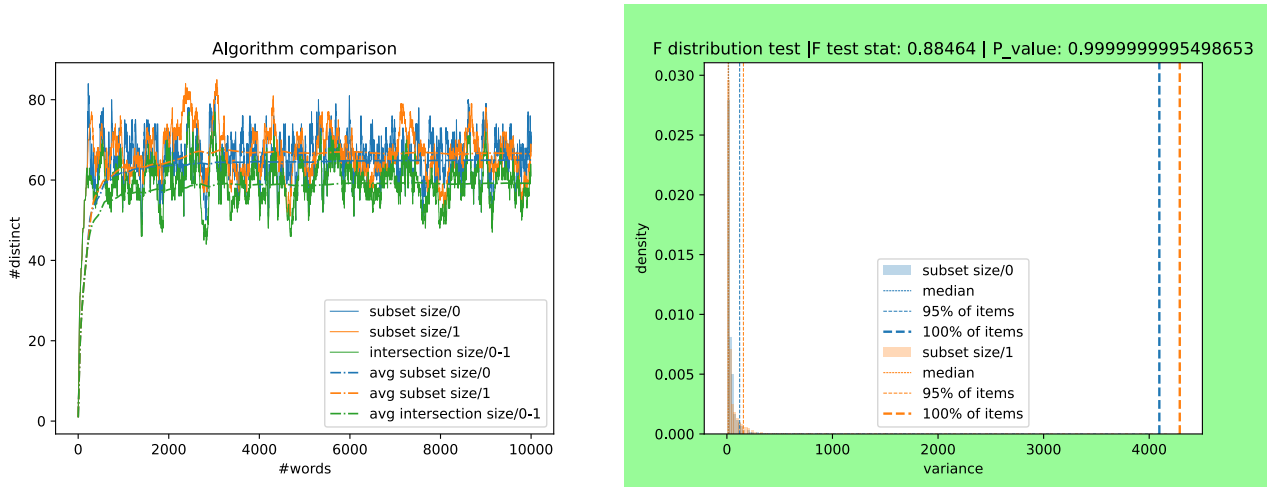


Figure 22: 0.h5

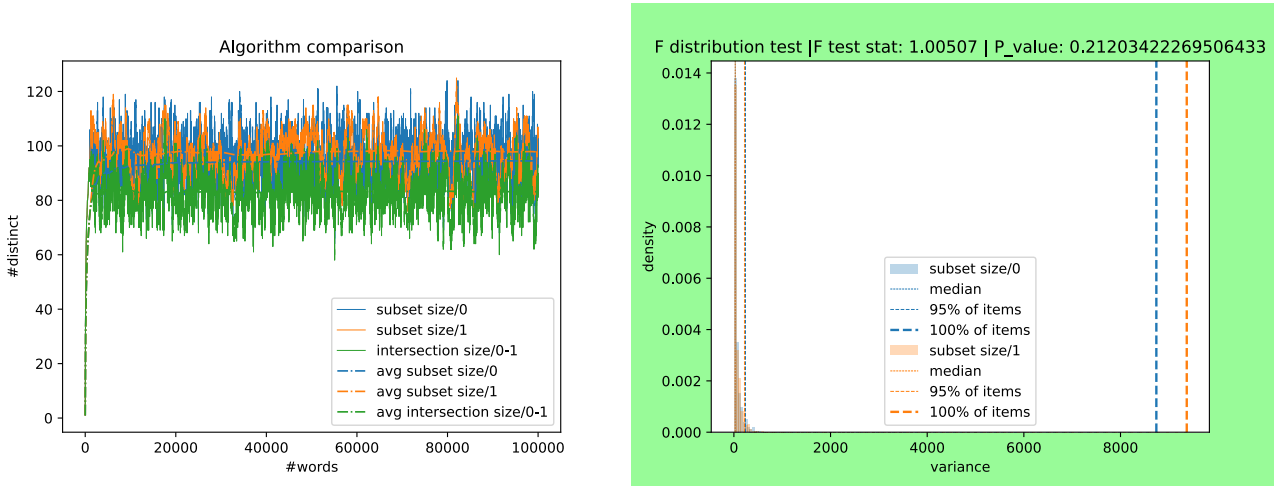


Figure 23: 1.h5

20.3 Experiment Certificates

20.3.1 Test 1


```
{
    "_epochs": {
        "_generator_factory": [
            [
                "EpochIter",
                {
                    "_function_key": "linspace",
                    "_function_params": [
                        1.5,
                        2.5,
                        1
                    ],
                    "_path_to_value": "/_evaluator_parameters/_generator/_builder_catalog/ZipfGenerator/_beta"
                }
            ]
        ]
    },
    "_evaluator_parameters": {
        "_algorithms": {
            "_builder_catalog": [
                [
                    "AffirmativeSamplingBuffered",
                    {
                        "_affirmative_sampling_creation_params": {
                            "_chosen_max_size": 20,
                            "_hash_function": {
                                "_builder_catalog": [
                                    [
                                        "TestHash",
                                        {
                                            "_seed": 0
                                        }
                                    ]
                                ]
                            }
                        },
                        "_buffer_size": 200
                    }
                ],
                [
                    "AffirmativeSamplingWindowed",
                    {
                        "_chosen_max_size": 20,
                        "_hash_function": {
                            "_builder_catalog": [
                                [
                                    "TestHash",
                                    {
                                        "_seed": 0
                                    }
                                ]
                            ]
                        },
                        "_window_size": 200
                    }
                ]
            ]
        },
        "_generator": {
            "_builder_catalog": [
                [
                    "ZipfGenerator",
                    {
                        "_beta": 1.5,
                        "_ndistinct": 500000,
                        "_rank_offset_left": 1000,
                        "_rank_offset_right": 0,
                        "_seed": 0
                    }
                ]
            ]
        },
        "_n_total_words": 10000,
```

```
        "_properties": [
            [
                "Property",
                {
                    "_data_type_key": "int",
                    "_function_key": "subset size",
                    "_sampling_frequency": 1,
                    "_which_algorithms": [
                        0,
                        1
                    ]
                }
            ],
            [
                "Property",
                {
                    "_data_type_key": "int",
                    "_function_key": "intersection size",
                    "_sampling_frequency": 1,
                    "_which_algorithms": [
                        [
                            0,
                            1
                        ]
                    ]
                }
            ]
        ]
    }
}
```

20.3.2 Test 2


```
{
    "_epochs": {
        "_generator_factory": [
            [
                "EpochIter",
                {
                    "_function_key": "linspace",
                    "_function_params": [
                        1.5,
                        2.5,
                        3
                    ],
                    "_path_to_value": "/_evaluator_parameters/_generator/_builder_catalog/ZipfGenerator/_beta"
                }
            ],
            [
                "EpochIter",
                {
                    "_function_key": "range",
                    "_function_params": [
                        3
                    ],
                    "_path_to_value": "/_evaluator_parameters/_generator/_builder_catalog/ZipfGenerator/_seed"
                }
            ]
        ]
    },
    "_evaluator_parameters": {
        "_algorithms": {
            "_builder_catalog": [
                [
                    "AffirmativeSamplingBuffered",
                    {
                        "_affirmative_sampling_creation_params": {
                            "_chosen_max_size": 20,
                            "_hash_function": {
                                "_builder_catalog": [
                                    [
                                        "TestHash",
                                        {
                                            "_seed": 0
                                        }
                                    ]
                                ]
                            }
                        },
                        "_buffer_size": 200
                    }
                ],
                [
                    "AffirmativeSamplingWindowed",
                    {
                        "_chosen_max_size": 20,
                        "_hash_function": {
                            "_builder_catalog": [
                                [
                                    "TestHash",
                                    {
                                        "_seed": 0
                                    }
                                ]
                            ]
                        },
                        "_window_size": 200
                    }
                ]
            ]
        },
        "_generator": {
            "_builder_catalog": [
                [
                    "ZipfGenerator",
                    {
                        "_beta": 1.5,
                        "_ndistinct": 500000,
```

```
                    "_rank_offset_left": 1000,
                    "_rank_offset_right": 0,
                    "_seed": 0
                }
            ]
        ]
    },
    "_n_total_words": 10000,
    "_properties": [
        [
            "Property",
            {
                "_data_type_key": "int",
                "_function_key": "subset size",
                "_sampling_frequency": 1,
                "_which_algorithms": [
                    0,
                    1
                ]
            }
        ],
        [
            "Property",
            {
                "_data_type_key": "int",
                "_function_key": "intersection size",
                "_sampling_frequency": 1,
                "_which_algorithms": [
                    [
                        0,
                        1
                    ]
                ]
            }
        ]
    ]
}
}
```

20.3.3 Test 3


```
{
    "_epochs": {
        "_generator_factory": [
            [
                "EpochIter",
                {
                    "_function_key": "linspace",
                    "_function_params": [
                        1.5,
                        2.5,
                        3
                    ],
                    "_path_to_value": "/_evaluator_parameters/_generator/_builder_catalog/ZipfGenerator/_beta"
                }
            ],
            [
                "EpochIter",
                {
                    "_function_key": "range",
                    "_function_params": [
                        3
                    ],
                    "_path_to_value": "/_evaluator_parameters/_generator/_builder_catalog/ZipfGenerator/_seed"
                }
            ]
        ]
    },
    "_evaluator_parameters": {
        "_algorithms": {
            "_builder_catalog": [
                [
                    "AffirmativeSamplingBuffered",
                    {
                        "_affirmative_sampling_creation_params": {
                            "_chosen_max_size": 20,
                            "_hash_function": {
                                "_builder_catalog": [
                                    [
                                        "TestHash",
                                        {
                                            "_seed": 0
                                        }
                                    ]
                                ]
                            }
                        },
                        "_buffer_size": 200
                    }
                ],
                [
                    "AffirmativeSamplingWindowed",
                    {
                        "_chosen_max_size": 20,
                        "_hash_function": {
                            "_builder_catalog": [
                                [
                                    "TestHash",
                                    {
                                        "_seed": 0
                                    }
                                ]
                            ]
                        },
                        "_window_size": 200
                    }
                ]
            ]
        },
        "_generator": {
            "_builder_catalog": [
                [
                    "ZipfGenerator",
                    {
                        "_beta": 1.5,
                        "_ndistinct": 500000,
```

```
                    "_rank_offset_left": 1000,
                    "_rank_offset_right": 0,
                    "_seed": 0
                }
            ]
        ]
    },
    "_n_total_words": 30000,
    "_properties": [
        [
            "Property",
            {
                "_data_type_key": "int",
                "_function_key": "subset size",
                "_sampling_frequency": 1,
                "_which_algorithms": [
                    0,
                    1
                ]
            }
        ],
        [
            "Property",
            {
                "_data_type_key": "int",
                "_function_key": "intersection size",
                "_sampling_frequency": 1,
                "_which_algorithms": [
                    [
                        0,
                        1
                    ]
                ]
            }
        ]
    ]
}
}
```

20.3.4 Test 4


```
{
    "_epochs": {
        "_generator_factory": [
            [
                "EpochIter",
                {
                    "_function_key": "range",
                    "_function_params": [
                        2
                    ],
                    "_path_to_value": "/_evaluator_parameters/_generator/_builder_catalog/ZipfGenerator/_seed"
                }
            ]
        ]
    },
    "_evaluator_parameters": {
        "_algorithms": {
            "_builder_catalog": [
                [
                    "AffirmativeSamplingBuffered",
                    {
                        "_affirmative_sampling_creation_params": {
                            "_chosen_max_size": 20,
                            "_hash_function": {
                                "_builder_catalog": [
                                    [
                                        "TestHash",
                                        {
                                            "_seed": 0
                                        }
                                    ]
                                ]
                            }
                        },
                        "_buffer_size": 1000
                    }
                ],
                [
                    "AffirmativeSamplingWindowed",
                    {
                        "_chosen_max_size": 20,
                        "_hash_function": {
                            "_builder_catalog": [
                                [
                                    "TestHash",
                                    {
                                        "_seed": 0
                                    }
                                ]
                            ]
                        },
                        "_window_size": 1000
                    }
                ]
            ]
        },
        "_generator": {
            "_builder_catalog": [
                [
                    "ZipfGenerator",
                    {
                        "_beta": 2.0,
                        "_ndistinct": 500000,
                        "_rank_offset_left": 1000,
                        "_rank_offset_right": 0,
                        "_seed": 0
                    }
                ]
            ]
        },
        "_n_total_words": 100000,
        "_properties": [
            [
                "Property",
                {
```

```
                    "_data_type_key": "int",
                    "_function_key": "subset size",
                    "_sampling_frequency": 1,
                    "_which_algorithms": [
                        0,
                        1
                    ]
                }
            ],
            [
                "Property",
                {
                    "_data_type_key": "int",
                    "_function_key": "intersection size",
                    "_sampling_frequency": 1,
                    "_which_algorithms": [
                        [
                            0,
                            1
                        ]
                    ]
                }
            ]
        ]
    }
}
```

21 References

References

[1] Hdf5 for python. URL https://docs.h5py.org/en/stable/.

[2] Pep 8 – style guide for python code. URL https://www.python.org/dev/peps/pep-0008/.

[3] M. Anaya. *Clean Code in Python*. Packt Publishing, 2021. ISBN 9781800560215.

[4] B. Efron and T. Hastie. *Computer Age Statistical Inference*. Cambridge University Press, 2018. ISBN 9781107149892.

[5] R. Ferrer-i Cancho. Hidden communication aspects in the exponent of zipf's law. *Glottometrics*, 11:98–119, 2005. URL http://hdl.handle.net/2117/176181.

[6] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *Journal of computer and system sciences*, 31:182–209, June 13, 1984.

[7] A. Gakhov. *Probabilistic Data Structures and Algorithms for Big Data Applications*. Deutsche Nationalbibliografie, 2019. ISBN 9783748190486.

[8] M. Gorelick and I. Ozsvald. *High Performance Python*. O'Reilly, May 2020. ISBN 9781492055020.

[9] O. W. in Data. Carbon dioxide emissions factor, kg co2 per mwh, 2021. URL https://ourworldindata.org/grapher/carbon-dioxide-emissions-factor.

[10] R. Johanson. *Numerical Python*. Apress, 2019. ISBN 9781800560215.

[11] C. Martínez. Probabilistic and combinatorial techniques in data stream analysis, 2021.

[12] S. J. Nielson and C. K. Monson. *Introduction to the Theory of Computation*. Deutsche Nationalbibliografie, 2013. ISBN 9781133187790.

[13] N. Ozimica. using requirements.txt to automatically install packages from conda channels and pip in a new conda environment. URL https://stackoverflow.com/questions/63379968/using-requirements-txt-to-automatically-install-packages-from-conda-channels-and.

[14] E. Parliament. Co2 emissions from cars: facts and figures (infographics), 18-04-2019. URL https://www.europarl.europa.eu/news/en/headlines/society/20190313ST031218/co2-emissions-from-cars-facts-and-figures-infographics.

[15] Statista. Precio medio final anual de la electricidad en españa de 2010 a 2021, 2021. URL https://es.statista.com/estadisticas/993787/precio-medio-final-de-la-electricidad-en-espana/.

[16] D. Tang. Cs240 – lecture notes: Hashing, March 2011. URL https://www.cpp.edu/~ftang/courses/CS240/lectures/hashing.htm.

[17] M. J. Wainwright. *High-Dimensional Statistics*. Cambridge University Press, 2019. ISBN 9781108498029.