Feature Selection using LAID and its Implementation on High-Performance Computing Systems

Paulo Morgado
Instituto Superior Técnico, Lisboa, Portugal
paulo.morgado@tecnico.ulisboa.pt

ABSTRACT

Of the many current challenges that the increasing availability of data poses to society and organizations, the ability to collect, store, process, analyze and extract knowledge in a helpful time frame stands out.

This dissertation focuses on the processing of highly dimensioned datasets, such as those generated by High-throughput sequencing (HTS) techniques, which are increasingly common in areas of knowledge such as bioinformatics.

Among the dimensionality reduction techniques, feature selection has become crucial since it reduces the high dimensionality of large datasets.

A possible approach to perform the feature selection that considerably reduces the dimension without increasing the inconsistency of the data is the use of Logical Analysis of Inconsistent Data (LAID). Several studies in recent years have demonstrated its potential in solving this problem and highlighted its advantages as a systematic methodology, robust, easy to interpret, and capable of dealing with inconsistent data [1], [2].

The same studies revealed processing times longer than desired for full utilization and pointed out the solution for executing the algorithms using parallel processing and mobilization of a high-performance computing (HPC) installation.

This work represents another contribution to this effort by addressing dataset storage methods suitable for parallel processing, algorithm parallelization solutions, high-performance environment configuration. Finally, it tests in the HPC environment of the Infraestrutura Nacional de Computação Distribuída (INCD). What allowed us to describe a fit-for-purpose dataset storage solution as well as parallel processing in an HPC environment can reduce processing times for the end-user, achieving a satisfactory 82% reduction relative to best sequential time.

KEYWORDS

Data mining, High dimensionality, Feature selection, Logical Analysis of Inconsistent Data (LAID), Parallel computing

1. Introduction

Our way of life is increasingly driven by data [3], and dependent on our capacity to collect, store, process, analyze and extract knowledge from a constantly increasing amount of data in a useful time. The same phenomenon can be observed in scientific environments [4]. The exponential growth of data generated by scientific experiments, instruments, and sensors, considering their volume, complexity, and even scale of distribution, has become a critical factor in several science disciplines, which induces the need to analyze ever-increasing volumes of data on time. Some authors identified this need as a new paradigm in scientific production and adopted the term coined by Jim Gray, "The Fourth Paradigm" [5]. According to them, Science started empirically, then theoretical, in the last few decades, computational and, recently, became data-driven. Another author [6], call this trend a "Data-intensive science", Whether in the scientific, business, or governmental sphere the same need led to the creation of new ways to manage distributed data, processing it, and mainly analyzing it to obtain information and useful knowledge.

This processing and analysis are usually called knowledge discovery from data or KDD and one essential stage [7] is known as Data mining that can be defined as the process of discovering patterns and extracting knowledge from large amounts of data [8]. Patterns on data is a generic expression that aggregates concepts like identifying groupings of data not previously known, detecting unusual records, or finding unknown dependencies. This process can be automatic or, very often, semi-automatic [9].

As it is often impracticable to analyze the entire Dataset, it is necessary to know how to choose which features the analysis should focus on, to the detriment of all others. It is on this scope that this work focuses, specifically in situations where this feature selection becomes essential, as in the case of highly dimensioned data sets, as the called Omics datasets. Omics is a neologism associated with biology suffixes such as genomics, transcriptomics, [1] and usually refers to biological parameters commonly called biomarkers.

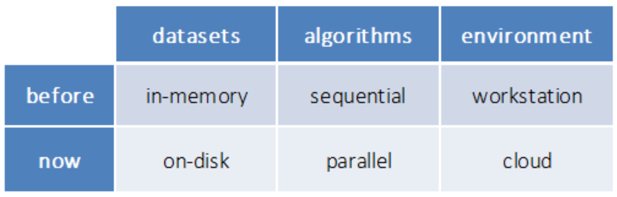
In this sense, dimensions reduction is the specialty in the scope of data mining, and one of the possible ways to accomplish it is through feature selection, which is mandatory on highly dimensioned datasets for which the analysis of all the dimensions is impractical and counterproductive. This task becomes even more necessary when the number of observations is low, the data matrix is sparse, and the analysis of row classification reveals inconsistencies. The problem addressed in the present work is an example of that. Considering that a feature selection problem with a dataset involving one million attributes was solved in [1], using partitions of the problem and a new method called Logical Analysis of Inconsistent Data (LAID) proposed by [10], this present research aims to continue this effort.

1.1. Motivation and scope

The 2018 experiment [1] uses a computer cluster running at the Portuguese National Infrastructure for Distributed Computing (INCD), for parallel processing while also using the dataset in memory. As a final remark, the authors consider three major aspects: data, algorithm, and computer environment. Since large datasets with many dimensions are becoming the new normal, they advocate that:

- In-memory data access should be replaced by on-disk data access.
- The new algorithms need low time complexity and problem decomposition to allow parallelizing the problem. Therefore, the intense use of computationally expensive algorithms, such as metaheuristics, should be reduced.
- And last, the desired environment should be in the cloud, running the programs in parallel in High-Performance Computing (HPC).

This is in line with the new trends for compute-intensive algorithms [11]. and constitutes a paradigm shift in scientific environments, [1], illustrated in Figure 1.



	datasets	algorithms	environment
before	in-memory	sequential	workstation
now	on-disk	parallel	cloud

Figure 1 - The paradigm shift in scientific environments

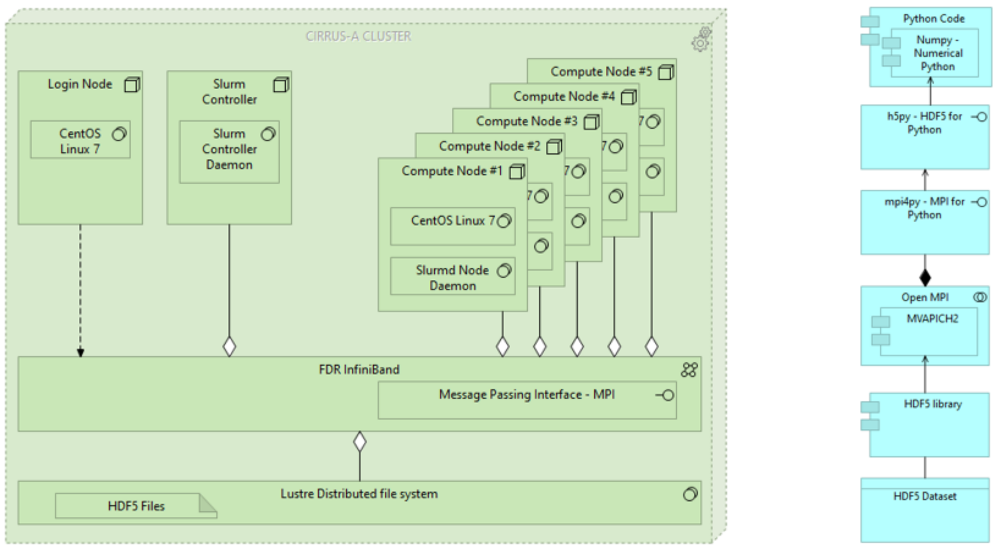
Later [2] revisited this problem using LAID and a different computational approach: sequential processing on a single machine and the dataset stored in high-performance data management and storage suite (HDF5) on disk. Both experiments concluded that the high processing time remains an additional problem that limits its application. The challenge and motivation for this new approach to the described problem are to reduce the processing time, using the computational paradigm: parallel processing, on-disk data access, and cloud/HPC environment.

The purpose of this research following a Design Science Research Methodology (DSRM), with a Design & Development centered approach, is to discover how parallel processing can reduce processing times for the end-user and describe how this new computational paradigm applies to this problem. The remainder of this document is structured around the mentioned tree concepts, Section 2 presents the environment, Section 3 introduces the dataset used and Storage strategy on HDF5 format, Section 4 describes the parallelization of the LAID algorithms, Section 5 presents the results of the experiment performed and Section 6 presents the conclusions.

2. The HPC environment on INCD cloud

HPC, also known as supercomputing, refers to computing systems with extremely high computational power that can solve hugely complex and demanding problems [12],[13], performance is commonly measured in floating-point operations per second (FLOPS).

INCD is a digital infrastructure supporting research, provides computing and storage services to the Portuguese scientific and academic community. And kindly provided computational resources in the Cirrus-A HPC cluster, that allowed the execution and testing of the code that constitutes the artifact of this work. On this cluster, there are 5 compute nodes AMD EPYC 7501 available, running CentOS 7, with 64 cores per node, interconnected by FDR InfiniBand 56Gbps [14]. The INCD filesystems are based on the Lustre distributed filesystem. Figure 2a) presents a view of the Technology usage of the Cirrus-A cluster and the main components used. The highlight for components essentials for this work such as HDF5 a data model, library, and file format for storing and managing data. [15] And the Message-Passing Interface (MPI), [16], addresses the message-passing parallel programming model and MPI-I/O that provide routines for file manipulation and data access by multiple processes. Figure 2b) presents an application cooperation diagram of the full component stack for parallelization. A significant part of the choice of software modules and versions used resulted from the excellent support and knowledge of the team that manages this facility.



a) Technology usage of Cirrus-A HPC Cluster and components used

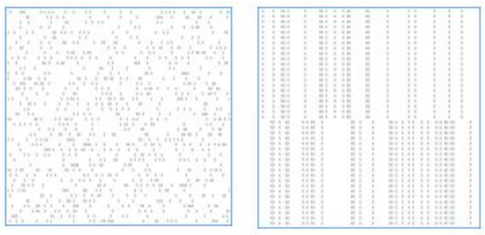
b) Application cooperation

Figure 2 - Technology usage of Cirrus-A HPC Cluster and Application cooperation

3. The Dataset, Storage strategy on HDF5 format

3.1. The Dataset used

The Source-Dataset used contains a total of 2k observations x 1000k features of synthetic data representing Omic data, all observations are labeled according to binary classification. was received in the form of text files. So, a python program was developed to extract data from text files and load them into the dataset in the HDF5 file.



a) random pattern

b) organized pattern

Figure 3 - Sample of distributions patterns found on Source-Dataset

After the dataset loaded in HDF5, an exploratory analysis was performed, allowing us to conclude that only 19.57% of the cells contain a value, which means a sparse matrix. Also, the sparsity is not uniform across the columns, because there are two different distribution patterns interspersed between blocks of 100k columns where a random distribution pattern a) alternates with a more concentrated pattern b), and in the latter, 31% of the columns do not contain any value.

3.2. HDF5 Storage Layout: Chunking Analyses

The default storage layout of HDF5 files is contiguous storage, "data of a multidimensional array is serialized along the fastest-changing dimension and is stored as a contiguous block in the file. This storage mechanism is recommended if the size of a dataset is known and the storage size for the dataset is acceptable to the user" [17]. Datasets may also be created using HDF5's chunked storage layout. This means the dataset is divided up into regularly sized pieces which are stored haphazardly on disk and indexed using a B-tree. This storage also makes it possible to resize datasets. For these reasons, choosing the storage layout and internal organization of the HDF5 file requires analysis and testing to allow for conclusions. To perform it, a test protocol was defined considering three subsets of the source dataset, based on the number of columns: L, XL, and XXL indicated in Table 1. The different options tested are described in

Table 2. An additional explanation is needed as to "Best Fit chunk" which depends on prior knowledge of the read cache size defined at HDF5 installation.

Table 1 - Datasets used in the read/write test protocol

Dataset	Rows x Columns.	GB
200K (L)	2k x 200k	0,37
600K (XL)	2k x 600k	1,12
1000K (XXL)	2k x 1000k	1,86

Table 2 - Chunking layout tested

Layout	Meaning
Row chunk	Equals the dimension of the entire row, regardless of its dimension
Best-Fit chunk	Is a chunk whose dimension perfectly fits the defined HDF5's read cache size
Contiguous	Don't use chunks
Auto-chunk	Gives HDF5 the responsibility for choosing the dimension

3.2.1. Write Operations over the HDF5 Source-Dataset

These tests were carried out at the expense of successive parallel write operations over the Source-Dataset and all times presented are the average of the runs performed. The noteworthy fact is that of the total data loading time, about 60% of the time is spent extracting the data from the text file, 40% transforming it into memory and a remarkable less than 0.5% is spent preserving the dataset in the HDF5 file. This is done in a single write operation which has proven to be extremely efficient regardless of the data block size and chunking layout.

It was also verified in these tests an excessive variability of the times between similar runs. the workload and I/O conditions of the cluster influenced the times obtained mainly in the L tests, even so, and given that in the portion concerning the write operation presented in Table 3, these are very low times, (all below 8 seconds), the focus has to be on small differences in seconds and here are two layouts that stand out negatively; Row chunk for datasets with less than 1000K columns and Auto-chunk with such poor performance that it was abandoned.

Table 3 - Average time to write an entire block of data on HDF5 file in seconds

Layout	200K (L)	600K (XL)	1000K (XXL)
Row chunk	4,91	2,08	1,82
Best Fit chunk	1,27	2,08	1,82
Contiguous	3,87	1,24	5,98
Auto-chunk	7,12		

Regarding the use of chunking in write operations, except for Auto-Chunk, there are no major discrepancies on observed times considering the dataset size and the chunking used. Not using chunking at all is not heavily penalized on write operations and may even be faster on 600K column datasets. Best Fit chunk is the most difficult to use and is probably the best solution for some sizes, where it is easier to adjust the dimension to the HDF5 cache size.

3.2.2 Read Operations over the HDF5 Source-Dataset

For this research, the storage layout must be aligned with the dataset access patterns, that is, the read operations, which are more relevant than the analysis to the write operations. Hence the choice to perform this test using the code from the 2nd step of the LAID methodology, in this step, access to the Source-Dataset requires the complete reading of the line, as each line is compared with all others, therefore, this reading pattern is perfectly suited to the intended test. Figure 4 chart compares the time consumed reding over different HDF5 Storage layout options.

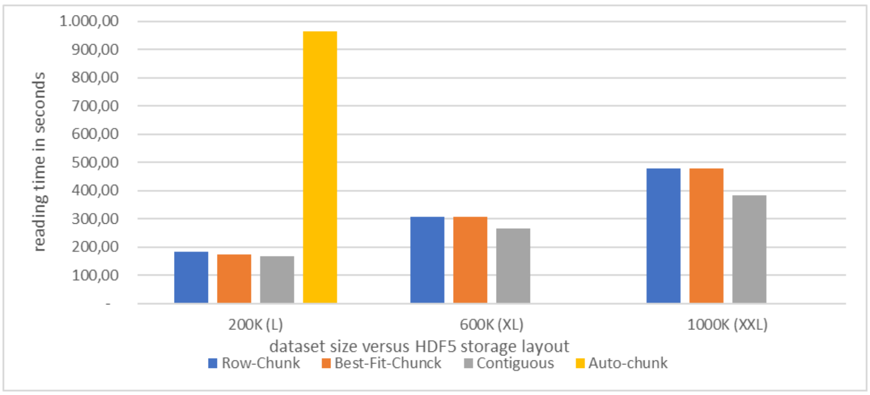


Figure 4 - Average time in seconds to read operations versus HDF5 Storage layout

About "Best Fit chunk" it practically matches the byte dimension of the XXL row, it is also possible to make a reasonable match with the L using 5 rows per chunk. but XL is penalized as it wastes almost half the cache value on each read. Still and except for the Auto-chunk option, which performs so badly even on a small dataset that was abandoned for the larger ones, the times between the remaining two chunking options are relatively similar.

Conclusions can be drawn from this reading pattern, chunking is an option to be considered in no way an obligation. Even for the best chunking option, given the read cache size defined in the HDF5 installation and the read pattern used by the algorithm, it is observed that the contiguous storage layout systematically allows greater speed and access. This speed is hardly distinctive in smaller datasets (L) but gradually becomes more significant.

From all the above, the preferred layout strategy for Source-Dataset is not to use chunking at all. Also, for the DM-Dataset obtained in the 2nd step of LAID, because it is accessed via row and column, using any form of chunking is an error.

4. Parallel LAID

4.1. LAID methodology

This method arises from the combination of Rough Sets Theory [18] and Logical Analysis of Data (LAD) methodology [19]. Both approaches are a subset of filter models, which goal is to reduce the number of dataset attributes using two phases: problem transformation and optimization. Their specificity is to keep the semantics of the data by removing only the redundant data.

Logical Analysis of Inconsistent Data (LAID) was proposed in 2011 by [10] in the scope of a Paremiologic Study and described by the authors as a blend of the best qualities of each of former methodologies, being able to deal with inconsistent data and non-dichotomized classes, characteristics of Rough Sets, [20] as well as the computational effectiveness of LAD. Also allows integer attributes, with associated costs. [21] which allows for a greater range of selection strategies. A detailed description with examples can be found in [22].

4.3. Coding the LAID algorithms

Given the prior availability of a codebase for the LAID algorithms written in the C and Python programming languages, from previous work [1] and [2], these source codes were used here with an emphasis on the Python version that already used HDF5, so part of this work consisted of adapting the sequential code and in the further developments needed, such as code improvements, changes in the execution flow, the introduction of a configuration file, different use of files, and HDFS datasets, and introduction of techniques aimed at parallelization.

According to [1],[2], the Logical Analysis of Inconsistent Data Algorithm consists of the following steps:

- Input: dataset D = {O, X∪C} with binary variables
- Output: (number of features, accuracy)
- check data inconsistencies and add dummy variable "jnsq" as a discriminant feature to remove any inconsistency in addition check and remove all redundant observations.
- disjoint matrix generation [Ai,j] this is a O(n2) in the Big O notation
- number of features = Minimum Set Covering Problem. In this third step of LAID, a heuristic is applied to solve the set covering problem. This is an O(log n) problem.
- accuracy = Cross-validation

Step 1 is fast, even without parallelization, and the algorithms are complex to parallelize, so the following was given priority and not covered in this work. Step 4 consists of validating the obtained solution, so it does not require parallelization at all.

Step 2 - Disjoint Matrix Generation

This step consists of reading all the Source-Dataset's rows and comparing them with all those that have a different class value, the disjoint rows found are written in a destination dataset, which for operational reasons has been decided to be stored in a separate HDF5 file, this new derived dataset has been named DM-Dataset.

Because there are no dependencies between each iteration of the algorithm used, i.e., each observation of the Source-Dataset is treated independently, its parallelization appears to be simply requiring only a few adaptations to the serial code. Technically it is only necessary to distribute the rows of the source-Dataset by the processors assigned to the run, in a horizontal data partition. However, it was not so simple. All parallel tasks on the run write the generated rows to the same HDF5 dataset. If this recording could be sequential, something like adding new rows to the dataset, the process would be simple, but it can't, the HDF5 datasets used are all fixed length. This implies, to avoid overwriting, strictly controlling, and segregation of the write-row index, as is done for reading.

It is obviously possible to define a formula to calculate rows index, but only if the Source-Dataset does not have redundant rows and if it is sorted by the value of the class(es) and this effectively is. But this sort, despite good practice, is not a requirement of LAID and it cannot be considered an assumption. Which makes this an unsolvable problem, because before the execution of the algorithm it is not possible to know how many, and in which index the rows should be written. This insolubility inspired the solution, not elegant but effective, the algorithm will run twice. In the first pass, it identifies the source and destination rows, without spending time writing to disk. With this valuable information, the second pass knows exactly what and where to write. The bet is that parallelization allows not only to recover the time spent on the first pass but also to save more time because it avoids unnecessary comparisons.

The code was also changed to include a buffer, which allowed the number of write operations to be reduced by writing more data at a time. The introduction of this buffer has proved so useful that even the serial version benefits from it, (29% from the initial time).

Step 3 – Find a solution with the minimum set covering algorithm

This is the bottleneck at all steps of the LAID methodology. The algorithm of this step is inherently serial. there is a clear dependence between interactions given that the computation performed in any iteration depends on the result of the previous iteration. The approach followed was to perform the vertical partition of the DM-Dataset columns and their distribution by parallel tasks. But the different solutions found must be compared and the best one chosen. which requires communication and cooperation between processors/parallel tasks, so, in this work, the approach for a parallel solution was limited to the first iteration. For this reason, the goal of parallelizing this step was suspended. Literature reveals various other potential parallel solutions [23],[24]. There is a common point in all of them relying on some form of communication between processors. In conclusion, the next approach will have to make use of MPI point-to-point or collective communication modes via shared-memory or external memory that allows coordination between parallel tasks/processors.

4.4. Performance Metrics for Parallel Systems

Whereas the main objective of parallel computing is increasing speed and reducing time complexity is necessary to measure the result. for this purpose, two performance metrics are usually used [1],[25], to measure the efficiency of an algorithm in terms of the time complexity factor. N computers working simultaneously can increase the speed up to N times, or an approximation of N as other factors can influence the time spent, such as bottlenecks communications. Parallel execution (run) time is usually defined as the time that elapses from the moment a parallel calculation starts until the moment the last processor finishes execution. The Speedup Sp factor is defined as the ratio of the serial runtime of the best sequential algorithm for solving a problem to the time taken by the parallel algorithm to solve the same problem on p processors.

$$S_p = \frac{\text{Execution time using one processor}}{\text{Execution time using } p \text{ processors}} = \frac{t_s}{t_p} \qquad (1)$$

Where ts (Tserial) is the execution time on a single processor or serial run time. tp (Tparallel) is the execution time on a multiprocessor or parallel run time and Sp is the increase in speed by using multiprocessor.

Efficiency E is defined as the ratio of speedup Sp to the number of processors P. Efficiency measures the fraction of time for which a processor is usefully utilized.

$$E = \frac{\text{Speedup } S_p}{\text{number of processors } P} \qquad (2)$$

5. Computational Experiments and Results

All Python code has been provided with performance counters for benchmarking whose information was cross-referenced with the start and end date-time of each job. therefore, there is high confidence in the observed time, however, some caution is needed in analyzing the reported times. As referred above, on several occasions, the same code and data running in different periods return the same result but with discrepancies in the time spent which indicates that the job environment is not fully isolated and is affected by cluster operational conditions probably associated with the distributed file system.

5.1. Serial, Hybrid, and Parallel Runs

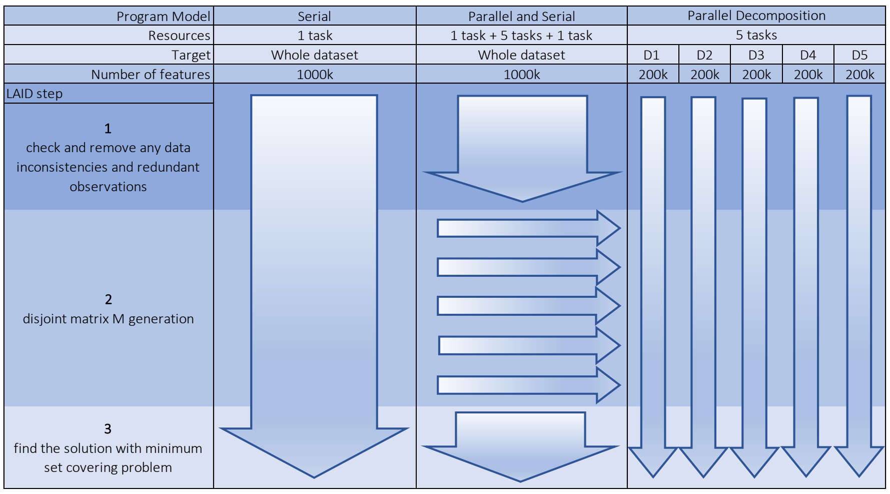


Figure 5 - Serial, Hybrid, and Parallel Runs

Figure 5 synthesizes the three forms of runs used in this work. The simplest model follows the sequential execution, is represented on the left side of the figure, the center represents a hybrid form, where part of the processing is serial, and part parallelized, and the parallel model is represented on the right side.

5.2. Serial Program Model

This solution is largely a repetition of the 2019 procedure [2] and mainly serves as a reference for current parallelization experiments. The average time consumed by each step is described in Table 4 This is approximately half the time reported by [2], however, there are too many differences between the two experiments, different datasets, and environments, the HPC effect, etc. to draw secure conclusions.

Table 4 – Average time spent running Serial LAID step by step

	LAID Step		T0
1	Check and remove inconsistency and redundant observations		1.254,96
2	Disjoint matrix generation	first-pass	N/A
		second-pass	1.352,21
3	Find solution with minimum set covering problem		6.221,88
		total in seconds:	8.829,06
		total in minutes:	147,15

5.3. Hybrid of Serial and Parallel Programming Models

In this solution, only step 2, DM-Dataset generation, is parallelized through a horizontal data partition. For this specific dataset, 5 parallel tasks were used so, the 1700 lines were divided by 5 and each job perform 340x300 comparisons with the results described in Figure 6. As expected, for the parallel version the difference is appreciable, a speedup of 3,04 with an efficiency of 0,6, this value can even be further improved by fine-tuning the buffer size.

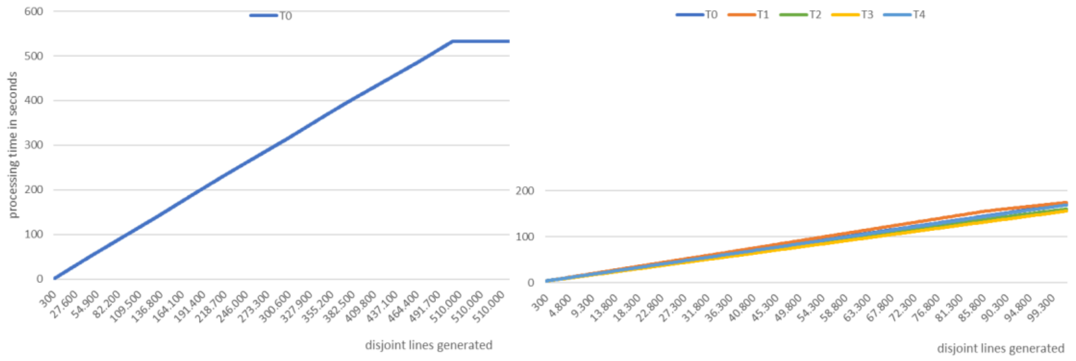


Figure 6 - DM-Dataset generation, Comparison of the serial version against 5 parallel tasks

The previous solution, although faster than the serial solution, is not practical. A much better solution is to run several jobs in parallel of all the serial code, that is, parallelize the execution of the serial version. This way, each job will process subsets of the entire dataset, for example, 200k column blocks. Then, after the parallel operation, a final run could be performed to reduce the result of the various solutions found. This way the bottleneck constituted by step 3, which was not fully resolved, has less impact.

Table 5 - Time spent running Parallel Decomposition

	LAID Step		T0	T1	T2	T3	T4
1	Check and remove inconsistency and redundant observations		32,28	32,60	32,81	33,11	32,98
2	Disjoint matrix generation	first-pass	211,71	212,91	206,97	211,94	212,05
		second-pass	237,44	240,05	236,69	242,38	237,51
3	Find solution with minimum set covering problem		1.044,77	988,40	956,70	1.068,36	918,17
		total in seconds:	1.526,19	1.473,95	1.433,17	1.555,79	1.400,71
		total in minutes:	25,44	24,57	23,89	**25,93**	23,35

Comparing and evaluating the running times of LAID algorithms for 1000k features using Serial against parallel version, where the same number of features is subdivided into 5 batches of 200k takes 147.15 minutes for serial to solve against 25.93 minutes on parallel decomposition a Speedup of 5,67.

5.5. Discussion

Table 6 presents a comparative board of the three approaches described above, for the last run parallel T1 was chosen because it has the worst (longest) performance of the third run. Some caution is needed when comparing this outcome as parallel T1 could not be directly compared with others as only handled 200k columns. Only the total time could be compared with no restrictions and as expected the fully parallel version is much faster than any of the others.

Table 6 - Comparative board of three approaches Serial/Hybrid/Parallel decomposition

	LAID Step		Serial	Hybrid	Parallel T3
1	Check and remove inconsistency and redundant observations		1.254,96	1.254,96	33,11
2	Disjoint matrix generation	first-pass	N/A	407,83	211,94
		second-pass	1.352,21	444,43	242,38
3	Find solution with minimum set covering problem		6.221,88	6.221,88	1.068,36
		total in seconds:	8.829,06	8.329,11	1.555,79
		total in minutes:	147,15	138,82	**25,93**

The result of the solution considering all columns of the source dataset is only the column with index 100004. The reason is that this column is exactly equal to the class column. Any quality evaluation would reveal 100% of accuracy and then the decision was for this specific case to skip the evaluation phase.

6. Conclusions and Future work

The research work is part of a sequence of previous works produced by different authors, to contribute to the usage of the Logical Analysis of Inconsistent Data methodology in the problem of selecting features in highly dimensioned datasets. A contribution that constitutes yet another "brick" in the much broader construction of knowledge for the effective solution of a current and increasingly relevant problem.

The approach followed is in line with current trends towards computationally intensive algorithms, which are increasingly considered a paradigm shift in scientific environments, where working data is stored on HDF5 format on disk and running the algorithms in parallel in a High-Performance Computing environment on the cloud.

The objective of discovering how parallel processing can reduce processing times for the end-user and describing how the HPC paradigm applies to this problem led to the decomposition of the problem into several parts that we tried to answer. To do this the Design Science Research methodology was followed and a Systematic Literature Review was carried out to compile a theoretical background that supports this research.

The key constituent parts of the HPC environment used have been described and documented, for future use.

Also, about the sample dataset used, useful information and data insights were extracted that can be useful in future situations to which is added a work of analysis and comparison of the best strategies and layouts for storing information in HDF5, meeting the reading patterns required by the different steps of LAID.

It has been fully confirmed and demonstrated that HDF5 is an excellent way to store the dataset used in a LAID analysis. It presents excellent performance for reading and writing. But even so, and especially in parallel use, it was also demonstrated that parallel code must implement ways to avoid overload to avoid degrading performance. The continuous layout was also indicated as the most suitable for the datasets used and the conditions where the alternatives can be used were described.

As for the parallelization of the LAID algorithms, code developed in previous works was adapted and newly developed as needed. This was later tested on the INCD HPC environment, following a testing protocol to produce the results presented. It deserves to be highlighted the reduction from 140 minutes of the serial version to 24 minutes of the parallel version of a dataset with 1000k features, which represents an acceptable time and an 82% reduction of the previous time.

Yet regarding this topic of parallelization, not all goals were achieved. It was left to do the parallelization step by step, but still, even in this case, it was possible to describe what is believed to be the solution which will certainly go by using the MPI collaborative mode.

To the constituents of the proposed artifact, which as stated is the parallel code, are the organization of HDF5 files and datasets and the orchestration of the process, it was also added an interesting set of lessons learned and guides that enrich it and justify the conviction about its usefulness.

It would be very interesting to continue the parallelization effort of the LAID steps. Using the MPI Collective Communication Mode and how this use impacts the execution of LAID algorithms, with an obvious emphasis on LAID's step 3. It would also be interesting to search for a fast and efficient way to remove irrelevant columns.

ACKNOWLEDGMENTS
This work was produced with the support of Infraestrutura Nacional de Computação Distribuída - INCD [National Infrastructure for Distributed Computation] funded by FCT and FEDER under the project 01/SAICT/2016 nº 022153.

REFERENCES

[1] L. Cavique, A. B. Mendes, H. F. M. C. Martiniano, and L. Correia, "A biobjective feature selection algorithm for large omics datasets," in *Expert Systems*, 2018, vol. 35, no. 4, doi: 10.1111/exsy.12301.
[2] J. Apolónia and L. Cavique, "Seleção de Atributos de Dados Inconsistentes em ambiente HDF5 + Python na cloud INCD," *Rev. Ciências da Comput.*, no. 14, pp. 85–112, 2019, doi: 10.34627/rcc.v14i0.
[3] A. Pentland, "The data-driven society," *Sci. Am.*, vol. 309, no. 4, pp. 78–83, 2013, doi: 10.1038/scientificamerican1013-78.
[4] J. Zhang, W. Wang, F. Xia, Y. R. Lin, and H. Tong, "Data-Driven Computational Social Science: A Survey," *Big Data Res.*, vol. 21, p. 100145, 2020, doi: 10.1016/j.bdr.2020.100145.
[5] T. Hey, S. Tansley, and K. Tolle, *The Fourth Paradigm*, 2009th ed. Microsoft Research, 2009.
[6] A. Agrawal and A. Choudhary, "Perspective: Materials informatics and big data: Realization of the 'fourth paradigm' of science in materials science," *APL Mater.*, vol. 4, no. 5, 2016, doi: 10.1063/1.4946894.
[7] A. Azevedo and M. F. Santos, "KDD, semma and CRISP-DM: A parallel overview," *IADIS Eur. Conf. Data Min.*, pp. 182–185, 2008, [Online]. Available: http://recipp.ipp.pt/handle/10400.22/136%0Ahttp://recipp.ipp.pt/bitstream/10400.22/136/3/KDD-CRISP-SEMMA.pdf.
[8] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*, Third Edit. Elsevier Inc., 2012.
[9] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical Machine Learning Tools and Techniques*, Fourth Edi. Elsevier Inc., 2017.
[10] L. Cavique, A. B. Mendes, and M. Funk, "Logical Analysis of Inconsistent Data (LAID) for a Paremiologic Study," 2011.
[11] D. Talia, "A view of programming scalable data analysis: from clouds to exascale," *J. Cloud Comput.*, vol. 8, no. 1, 2019, doi: 10.1186/s13677-019-0127-x.
[12] Ec.europa.eu, "High Performance Computing," *European Commission*, 2021. https://ec.europa.eu/digital-single-market/en/high-performance-computing (accessed Mar. 27, 2021).
[13] W. Zheng, "Research trend of large-scale supercomputers and applications from the TOP500 and Gordon Bell Prize," *Sci. China Inf. Sci.*, vol. 63, no. 7, pp. 1–14, 2020, doi: 10.1007/s11432-020-2861-0.
[14] INCD - Infraestrutura Nacional de Computação Distribuída, "INCD user documentation." https://wiki.incd.pt/books (accessed Apr. 03, 2021).
[15] The HDF Group, "HDF5 high performance data software library and file format." https://www.hdfgroup.org/solutions/hdf5 (accessed May 13, 2021).
[16] MPI-Forum, "MPI Standard." https://www.mpi-forum.org (accessed Jun. 18, 2021).
[17] Andrew Collette and contributors, "h5py - HDF5 for Python." https://docs.h5py.org/en/stable/index.html (accessed Jul. 12, 2021).
[18] Z. Pawlak, *Rough sets: Theoretical aspects of reasoning about data*, vol. 4, no. 5. Boston: Kluwer Academic Publishers, 1991.
[19] E. Boros, P. L. Hammer, T. Ibaraki, and A. Kogan, "Logical Analysis of Numerical Data," *Math. Program. Springer*, vol. 79, no. October, pp. 163–190, 1997.
[20] S. Rissino and G. Lambert-Torres, "Rough Set Theory - Fundamental Concepts, Principals, Data Extraction, and Applications," *Data Min. Knowl. Discov. Real Life Appl.*, no. February, 2009, doi: 10.5772/6440.
[21] L. Cavique, A. B. Mendes, M. Funk, and J. M. A. Santos, "A feature selection approach in the study of azorean proverbs," in *Exploring Innovative and Successful Applications of Soft Computing*, Exploring., Hershey: IGI Global, 2013, pp. 38–58.
[22] J. Apolónia and L. Cavique, "Seleção de atributos utilizando a Análise Lógica de Dados Inconsistentes (LAID)," *Repositório institucional da Universidade Aberta (UAb)*, 2019. http://hdl.handle.net/10400.2/8122 (accessed May 06, 2021).
[23] G. E. Blelloch, H. V. Simhadri, and K. Tangwongsan, "Parallel and I/O efficient set covering algorithms," *Annu. ACM Symp. Parallelism Algorithms Archit.*, pp. 82–90, 2012, doi: 10.1145/2312005.2312024.
[24] S. Chakravarty and A. Shekhawat, "Parallel and serial heuristics for the minimum set cover problem," *J. Supercomput.*, vol. 5, no. 4, pp. 331–345, 1992, doi: 10.1007/BF00127952.
[25] S. Rastogi and H. Zaheer, "Significance of parallel computation over serial computation," *Int. Conf. Electr. Electron. Optim. Tech. ICEEOT 2016*, pp. 2307–2310, 2016, doi: 10.1109/ICEEOT.2016.7755106.