**UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS**

Jonas Rossi Dourado

Delayed Transfer Entropy applied to Big Data

São Carlos

2018

Jonas Rossi Dourado

Delayed Transfer Entropy applied to Big Data¹

Dissertação apresentada à Escola de Engenharia de São Carlos da Universidade de São Paulo, para obtenção do título de Mestre em Ciências - Programa de Pós-Graduação em Engenharia Elétrica.

Área de concentração: Sistemas Dinâmicos

Orientador: Prof. Dr. Carlos Dias Maciel

São Carlos
2018

¹ Trata-se da versão corrigida da dissertação. A versão original se encontra disponível na EESC/USP que aloja o Programa de Pós-Graduação de Engenharia Elétrica.

AUTORIZO A REPRODUÇÃO TOTAL OU PARCIAL DESTE TRABALHO,
POR QUALQUER MEIO CONVENCIONAL OU ELETRÔNICO, PARA FINS
DE ESTUDO E PESQUISA, DESDE QUE CITADA A FONTE.

Ficha catalográfica elaborada pela Biblioteca Prof. Dr. Sérgio Rodrigues Fontes da
EESC/USP com os dados inseridos pelo(a) autor(a).

D726d	Dourado, Jonas Rossi Delayed Transfer Entropy applied to Big Data / Jonas Rossi Dourado; orientador Carlos Dias Maciel Maciel. São Carlos, 2018. Dissertação (Mestrado) - Programa de Pós-Graduação em Engenharia Elétrica e Área de Concentração em Sistemas Dinâmicos -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2018. 1. Delayed Transfer Entropy. 2. Parallelism Strategies. 3. Big Data Analysis. 4. Heterogeneous Computer Cluster. 5. Complex Systems. 6. Causality. I. Título.
-------	--

FOLHA DE JULGAMENTO

Candidato: Engenheiro **JONAS ROSSI DOURADO**.

Título da dissertação: "Delayed Transfer Entropy aplicado a Big Data".

Data da defesa: 30/11/2018.

Comissão Julgadora:

Resultado:

Prof. Associado **Carlos Dias Maciel**
(Orientador)
(Escola de Engenharia de São Carlos/EESC)

Aprovado

Prof. Titular **Alexandre Cláudio Botazzo Delbem**
(Instituto de Ciências Matemáticas e de Computação/ICMC)

Aprovado

Prof. Dr. **Ailton Akira Shinoda**
(Universidade Estadual Paulista "Júlio de Mesquita Filho"/UNESP – Ilha Solteira)

Aprovado

Decano do Coordenador do Programa de Pós-Graduação em Engenharia Elétrica:

Prof. Associado **Luís Fernando Costa Alberto**

Presidente da Comissão de Pós-Graduação:
Prof. Associado **Luís Fernando Costa Alberto**

Dedico este trabalho a minha noiva, meus irmãos e pais que sempre estiveram ao meu lado. Amo todos: Beatriz, Amália, Breno, Sarah, Ângela e Marcos
Dedico também aos meus Avós que muito me ensinaram sobre a vida.
E a todos que eu gostaria de passar mais tempo junto.

AGRADECIMENTOS

Agradeço antes de mais nada a Deus.

Agradeço meu amigo e orientador Carlos Dias Maciel por sempre estar disposto a ajudar.

Agradeço aos meus colegas de laboratório Daniel Lima, Fernando Santos, Giovana Nakashima, Jordão Oliveira Jr, Matheus Fogliatto, Michel Bessani, Rafael Ribeiro, Tadeu Gross, Talysson Santos, Victor Tsukahara e Willian Darwin Jr. As discussões científicas do dia a dia contribuíram para o meu desenvolvimento científico.

Aos meus amigos André Andrew Cunha, Bruno Brubru Begotti, Leandro Gaspar e Ivan Filgueiras que sempre incentivaram minha curiosidade.

Por último e não menos importante, agradeço a todos os meus amigos de José Bonifácio pela amizade, fazendo de lá, a melhor cidade de todas! Não tentarei listarei todos pois tenho certeza que um ou outro ficaria de fora injustamente.

“They did not know it was impossible so they did it”

Mark Twain

ABSTRACT

DOURADO, J. **Delayed Transfer Entropy applied to Big Data**. 2018. 97p.
Dissertação (Mestrado) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2018.

Recent popularization of technologies such as Smartphones, Wearables, Internet of Things, Social Networks and Video streaming increased data creation. Dealing with extensive data sets led the creation of term big data, often defined as when data volume, acquisition rate or representation demands nontraditional approaches to data analysis or requires horizontal scaling for data processing. Analysis is the most important Big Data phase, where it has the objective of extracting meaningful and often hidden information. One example of Big Data hidden information is causality, which can be inferred with delayed transfer entropy (DTE). Despite DTE wide applicability, it has a high demanding processing power which is aggravated with large datasets as those found in big data. This research optimized DTE performance and modified existing code to enable DTE execution on a computer cluster. With big data trend in sight, this results may enable bigger datasets analysis or better statistical evidence.

Keywords: Delayed Transfer Entropy. Parallelism Strategies. Big Data Analysis. Heterogeneous Computer Cluster. Causality. Surrogate.

RESUMO

DOURADO, J. **Delayed Transfer Entropy applied to Big Data**. 2018. 97p.
Dissertação (Mestrado) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2018.

A recente popularização de tecnologias como *Smartphones*, *Wearables*, Internet das Coisas, Redes Sociais e *streaming* de Video aumentou a criação de dados. A manipulação de grande quantidade de dados levou a criação do termo Big Data, muitas vezes definido como quando o volume, a taxa de aquisição ou a representação dos dados demanda abordagens não tradicionais para analisar ou requer uma escala horizontal para o processamento de dados. A análise é a etapa de Big Data mais importante, tendo como objetivo extrair informações relevantes e às vezes escondidas. Um exemplo de informação escondida é a causalidade, que pode ser inferida utilizando *delayed transfer entropy* (DTE). Apesar do DTE ter uma grande aplicabilidade, ele possui uma grande demanda computacional, esta última, é agravada devido a grandes bases de dados como as encontradas em Big Data. Essa pesquisa otimizou e modificou o código existente para permitir a execução de DTE em um *cluster* de computadores. Com a tendência de Big Data em vista, esse resultado pode permitir bancos de dados maiores ou melhores evidências estatísticas.

Palavras-chave: Delayed Transfer Entropy. Parallelism Strategies. Big Data Analysis. Cluster Heterogêneo de Computadores. Causality. Surrogate.

LIST OF FIGURES

Figure 1 – Data analysis and significance levels. From input data $x(t)$, it is estimated a series of surrogate signals $X_n^S(t)$, and for each of surrogate signal is determined time delayed transfer entropy. The $\mathbf{n}$ repetitions indicate a significance level from Equation 2.11 against random probabilities with the same power spectra and amplitude distribution as real data. Assuming the number of surrogate $\mathbf{n}$ equals three as shown in this figure, DTE Threshold line displayed on biggest ensemble plot has a significance level of 75%.	33
Figure 2 – Adapted from Santos, Maciel e Newland (2017). First a pre processing is executed on signals to remove non-linear trends on data. Then 10 surrogates are generated for each signal. Finally, DTE from original signal is normalized against average DTE from surrogate signals	34
Figure 3 – Beowulf cluster used in the experiments. Lines between devices may represent multiple Ethernet cables for diagram cleanness purpose. Cluster nodes are prefixed with "lps" following by identification number. Detailed cluster configuration is informed in Table 1.	38
Figure 4 – This graph show the resumed commit history. Note that although Part I was created temporally after Part IV , we did an prior <i>ad hoc</i> investigation. Its important to note that Part I measures only DTE performance in local mode.	40
Figure 5 – Component diagram showing DTE application (<i>DTEexec.py</i>) and DTE plot script (<i>DTEAnalysis.py</i>) relation to <i>lpslib</i> source files. <i>EmbeddingParams</i> method is responsible for finding Equation 2.9 delay between y^- and y^+	41
Figure 6 – DTE plot script matrix-like layout output example. Output file is in a vector format (Portable Document Format) with clear zoom visualization on electronic viewers. On bottom there is an artificially zoomed plot from permutation channel name 200 to channel name 27. On blue is the original signal DTE result, and other are DTE from surrogate signals, in this case, $nsur = 3$	45
Figure 7 – Input based profiling on DTE algorithm by varying maximum delay (<i>lag</i>) input	52
Figure 8 – Input based profiling on DTE algorithm by varying number of PDF discretization bins (<i>nbins</i>) input	53
Figure 9 – Input based profiling on DTE algorithm by varying number of PDF discretization bins (<i>nbins</i>) input for optimized CODE B	53

Figure 10 – Input based profiling on DTE algorithm by varying signal duration (<i>signalDuration</i>) input	54
Figure 11 – Code profile result on CODE B DTE algorithm.	55
Figure 12 – DTE inner loop execution time for both CODE A and CODE B (optimized DTE). Data measured with ten surrogate signals with 686053 samples each.	56
Figure 13 – CODE B Serial algorithm flowchart. One instance of this program was executed for each slice of an experiment list. Objects of this study are highlighted in blue.	57
Figure 14 – CODE C Data parallelism algorithm flowchart. Note that each dashed line means data transfer to every cluster node and every dotted line means data gathering and synchronization to host node. Objects of this study are highlighted in blue.	58
Figure 15 – CODE D Task parallelism algorithm flowchart. Note that each dashed line means data transfer to every cluster node and every dotted line means data gathering and synchronization to host node. Objects of this study are highlighted in blue.	59
Figure 16 – Data and Task Parallelism total execution time per number of surrogate signals. Point marks show numerical results in minutes. Lines show data fitted by linear square method	60
Figure 17 – DTE Task Parallelism performance comparison between different com- puting nodes to highlight cluster heterogeneity. In Task Parallelism, each experiment spawned <i>number of channels * (number of channels – 1)</i> DTE tasks of the same size across computing nodes, their execution times were used to calculate speedup of Y-axis computing node over X axis computing node and finally was made an average of calculated speedups for each cell. Execution log used to generate this plot was from 20 surrogates. Values are given in relative speedup	61
Figure 18 – Acceptance Letter from COMPLEXIS 2017	75

LIST OF TABLES

Table 1 – Cluster hardware configuration. RAM modules were list separately since some nodes have multiple memory modules to explore dual channel. Main storage describe storage media used during script execution, some nodes might have other unused storage media.	39
Table 2 – Cluster software configuration. Updated at shows the date when each cluster node was last fully updated.	41

LIST OF ALGORITHMS

1	DTE algorithm	43
2	IAAFT	44
3	Input based profiling	47

LIST OF ABBREVIATIONS AND ACRONYMS

DTE	Delayed Transfer Entropy
LHC	Large Hadron Collider
LPS	Signal Processing Laboratory
MI	Mutual Information
NIST	National Institute of Standards and Technology
PDF	Probability Density Function
TE	Transfer Entropy

CONTENTS

1	INTRODUCTION	23
2	THEORY	27
2.1	Big Data	27
2.1.1	Classification	27
2.1.2	Analysis	28
2.1.3	Performance issues	29
2.2	Delayed Transfer Entropy	30
2.2.1	Information Theory concepts	30
2.2.2	Delayed Transfer Entropy	32
2.2.3	Statistical Significance	33
3	MATERIAL AND METHODS	37
3.1	Hardware Setup	38
3.2	Software setup	39
3.3	DTE application	41
3.4	Part I - DTE time efficiency	45
3.5	Part II - Transfer Entropy serial optimization	47
3.6	Part III - DTE application cluster intrinsics	48
3.7	Part IV - DTE application cluster optimization	49
3.8	Part V - <i>lpslib</i> documentation	49
4	RESULTS AND DISCUSSION	51
4.1	Part I - DTE time efficiency	51
4.2	Part II - Transfer Entropy serial optimization	54
4.3	Part III - DTE application cluster intrinsics	56
4.4	Part IV - DTE application cluster optimization	58
4.5	Part V - <i>lpslib</i> documentation	62
5	CONCLUSION	63
	BIBLIOGRAPHY	65
	APPENDIX	73
	APPENDIX A – SUBMITTED PAPERS	75

A.1	COMPLEXIS 2017	75
A.2	Applied Soft Computing	77
A.3	Congresso Brasileiro de Automática 2018	79
A.4	Entropy journal	81
	APPENDIX B – PLOT PART II - TRANSFER ENTROPY SE- RIAL OPTIMIZATION	83
	APPENDIX C – DOCUMENTATION	85
C.1	Cluster installation	85
C.2	<i>ipyparallel</i> usage example	92
C.3	lpslib documentation	95

1 INTRODUCTION

Recent popularization of technologies such as Smartphones, Wearables, Internet of Things, Social Networks and Video streaming increased generated and collected data (HASHEM et al., 2015). Although extensive data creation is raising by general population, scientific research community already experienced it in many areas (LYNCH, 2008)(MARX, 2013), exemplified by CERN Large Hadron Collider (LHC) with 200 Petabytes (as of Jun/17) from particle collision experiments (GAILLARD, 2017) or European Bioinformatics Institute with 75 Petabytes (as of Dec/15) composed by DNA sequences, proteins, chemicals among other biology data (COOK et al., 2015).

Dealing with extensive data sets led the creation of term big data, with one definition by Hashem et al. (2015) as a set of technologies and techniques to discover hidden information from diverse, complex and massive scale datasets. Big data provide unique challenges in storage, transmission, acquisition, analysis and visualization (CHEN; ZHANG, 2014). According to Chen e Zhang (2014), the most critical challenge when dealing with the analysis is scalability, which researchers tackle by developing new algorithms or speeding up processors. Development of parallel computing is increased as data generation is scaling faster than processor speed (CHEN; ZHANG, 2014).

Song et al. (2017) strengthen analysis phase importance by stating that one of big data value perspective lies on analysis algorithms. A vital analysis phase aspect raised as a top priority by Chen e Zhang (2014) is timeliness, where the challenge is to give a prompt answer when selected data is large. Also, according to Hu et al. (2014), analysis phase is Big Data the most critical phase, where it has the objective of extracting meaningful and often hidden information (HASHEM et al., 2015).

One class of hidden information is causality, which Bareinboim e Pearl (2016) discuss and propose a framework to deal with commonly found big data biases such as confounding and sampling selection. Among the tools to infer causal relationships, there are Mutual Information used by Endo et al. (2015) to infer neuron connectivity and Granger causality used by Strohsal et al. (2015) to model causality between US and UK economies. Additionally, exist Transfer Entropy (TE), which allows identification of the cause-effect relationship by not accounting for straightforward and uniquely shared information (SCHREIBER, 2000). TE has been applied to many problems from diverse research fields e.g. finance (CAO; ZHANG; LI, 2017; YOOK et al., 2016); biosignals (MARZBAN-RAD et al., 2015); sensors (BERGER et al., 2016; HARTICH; BARATO; SEIFERT, 2016; ZHAI et al., 2016); complex networks (HARUNA; FUJIKI, 2016); climatology (HIRATA et al., 2016) and industrial energy consumption (YAO et al., 2017).

A derivation of TE metric called Delayed TE (DTE) is explored in [Kirst, Timme e Battaglia \(2016\)](#) to extract topology in complex networks, where they mentioned an *ad hoc* complex network example. Also, [\(WOLLSTADT et al., 2014\)](#) use DTE to calculate information transfer and delays from magnetoencephalography signals. Despite TE and DTE broad applicability, they have a high demanding processing power [\(SHAO et al., 2015\)](#), which is aggravated with large datasets as those found in big data. Many approaches were proposed to overcome performance issues such as an implementation using a GPU made by [\(WOLLSTADT et al., 2014\)](#) and implementation using an FPGA made by [\(SHAO et al., 2015\)](#).

Another known approach to speed up data analysis is parallelizing a program to run on a computer cluster. Parallel programs should be optimized to extract maximum performance from hardware architecture on a case by case, which is far from trivial according to [Booth, Kim e Rajamanickam \(2016\)](#). There exist different and combined manners to explore parallelism such as data parallelism and task parallelism [\(GORDON; THIES; AMARASINGHE, 2006\)](#). [\(CHOUDHURY et al., 2015\)](#) stated that choosing the configuration of parallel programs is a "mysterious art" in a study which they created a model aiming maximum speedup by balancing different parallelism strategies for both cluster and cloud computing environments.

The present research primary objective is to investigate alternatives to speedup DTE algorithm runtime in a big data context. The importance of proposed theme is reinforced by [Li, Qin e Yuan \(2016\)](#) conclusion that TE performance is about 3000 slower than Granger Causality, making TE unsuited for many Big Data analysis, a fact that is directly applied to derived DTE.

The idea behind this study emerged as a necessity brought by other Signal Processing Laboratory (LPS) projects. In particular, a currently going project on insect motor neurons network [\(SANTOS; MACIEL; NEWLAND, 2017; LIMA et al., 2016; ENDO et al., 2015\)](#) had a DTE performance demand to increase analysis confidence. Other LPS projects such as multi-scenarios Monte Carlo simulations [\(BESSANI et al., 2016\)](#) and optimization of large-scale systems reconfiguration [\(CAMILLO et al., 2016\)](#) also would be benefited from cluster configuration and usage know-how.

To accomplish research objective, we checked our DTE implementation time efficiency against existing literature, then, made serial performance optimizations to speedup DTE. After serial optimization, program code modifications were done to enable execution in a cluster environment. Finally, a study about different parallelism strategies was made to evaluate performance implications.

After researching about how to parallelize application and configure an *ipyparallel* Beowulf cluster, we found a lack of material with simple instructions. The lack of material and the possibility to help other colleagues in need to increase their experiment performance,

we proposed as a secondary objective creating an easy to follow guide with some examples about *ipyparallel* parallelization and a detailed manual on how to build an *ipyparallel*.

This document is organized as follows: Theory chapter present concepts necessary to comprehend this research; Material and Methods contains all steps to reproduce experiments divided by parts, with each part containing necessary described method and information for full reproducibility; Results and Discussion chapter is divided by each research part; In Conclusion, additionally, future works are suggested; Submitted papers are shown in [Appendix A](#).

2 THEORY

In this chapter, we start with an introduction to big data emphasizing its analysis performance issues. Afterward, we introduce basic Information Theory concepts necessary to Delayed Transfer Entropy. Finally, we discuss the importance of DTE performance with statistical significance in mind.

2.1 Big Data

There is a fragmented idea about what defines big data ([GANDOMI; HAIDER, 2015](#); [KITCHIN, 2014](#); [HU et al., 2014](#)) in the scientific community. The most prominent characteristic of big data is its size, with other common characteristics proposed by Doug Laney are explained by three Vs: Volume, Variety, and Velocity ([GANDOMI; HAIDER, 2015](#)). Volume is about the size; Variety refers to how data is stored (structured or not) and Velocity deals to how fast data is generated and analyzed. Industry perspective considers another "V" characteristics about big data, listed by Veracity, Variability, and Value ([HASHEM et al., 2015](#); [GANDOMI; HAIDER, 2015](#)). A big data definition pertinent to this study is given by National Institute of Standards and Technology (NIST) stating as where data volume, acquisition rate or representation demands nontraditional approaches to data analysis or requires horizontal scaling for data processing ([HU et al., 2014](#))

From an etymology standpoint, an investigation made by ([DIEBOLD, 2012](#)) affirmed that the term big data origin is intriguing and cannot be traced to a single person. Some of the pioneer users of big data term involve academy book ([WEISS; INDURKHYA, 1998](#)) and Silicon Graphics (SGI) former chief scientist Jonh Mashey as an industry representative.

2.1.1 Classification

Depending on author research focus, big data may have a different classification, such as dataset types ([SONG et al., 2017](#)); aspects ([HASHEM et al., 2015](#)) and system engineering phases ([HU et al., 2014](#)).

Aspect classification proposed by ([HASHEM et al., 2015](#)) allow performing additive labeling of big data. Listed aspects are:

a) Data Source;

Web & Social, Machine, Sensing, Transactions or IoT;

b) Content-Format;

Structured, Semi-structured or Unstructured;

c) Data Storage;

Document-oriented, Column-oriented, Graph based or Key-value;

d) Data Staging;

Cleaning, Normalization or Transform;

e) Data Processing;

Batch or Real time;

The review by (SONG et al., 2017) categorized big data looking at recent field advancements. The obtained categories are data types; storage models; analysis models; applications; data security and privacy. As collected datasets have different structures, sizes, density, and distribution (SONG et al., 2017), it led the division of data type into Online Network Data, Mobile and IoT Data, Geography Data, Spatial-Temporal Data, Streaming, and Real-time Data and Visual Data. An additional data type discussion by (SONG et al., 2017) was how each domain has its challenges and solving each of them may have a positive impact on future big data systems.

(HASHEM et al., 2015) and (SONG et al., 2017) have some overlap such as between Data Source and Data Types classification; Content-Format with Data Storage and Storage Models; Data Processing and applications. This overlap is expected as authors had a different objective in their reviews, with (HASHEM et al., 2015) more interested in cloud applications, while (SONG et al., 2017) was worried about big data analysis state of the art, challenges, and future researches.

Looking from a system engineering perspective, (HU et al., 2014) divided big data into four life cycle phases. Data generation is how data is generated. Data acquisition is the process of obtaining generated data from different sources and pre-processing to remove redundancies before saving it to a storage media. Data storage refers to store and manages data, including hardware and software levels. Finally, the Data Analysis phase is responsible for knowledge extraction.

2.1.2 Analysis

An extensively employed programming model to analyze big data is MapReduce (SONG et al., 2017), as seen in recent studies (CHANG, 2017; CHENG et al., 2017; SHANG; CHEN; YAN, 2017; SUN et al., 2016; ZHU; GE; SONG, 2017). The MapReduce abstraction was inspired by functional programming map and reduce functions, as explained in the seminal paper of Dean e Ghemawat (2008).

The main idea behind MapReduce is to compute intermediary results by applying a user-defined function with map operation and grouping them using reduce operation (HU et al., 2014). One key aspect of algorithm design using the MapReduce model is the balance between computation and communications, highlighted by Ullman (2012).

2.1.3 Performance issues

Software long run time is often an issue, especially in big data context (REYES-ORTIZ; ONETO; ANGUITA, 2015). To decrease software experiment execution time, one can use faster hardware or optimize underlining algorithms. Some hardware options to decrease execution time includes FPGA (AMEUR; SAKLY, 2017; MALDONADO; CASTILLO; MELIN, 2013), GPU (TING et al., 2016), faster processors (NASROL-LAHZADEH; KARIMIAN; MEHRAFSA, 2017) or computer clusters (BAZOW; HEINZ; STRICKLAND, 2017; KAPP; SABOURIN; MAUPIN, 2012). Algorithm optimizations examples are found in studies by Gou et al. (2017), Naderi et al. (2017) and Sánchez-Oro et al. (2017).

Among computer clusters, one well-known implementation is Beowulf cluster, made by connecting consumer grade computers on a local network using Ethernet or other suitable connection technology (YAO; CHANG; XIA, 2009). The term Beowulf cluster was coined by Sterling et al. (1995), which created the topology on NASA facilities as a low-cost alternative to expensive commercial vendor built High-Performance Clusters. Beowulf cluster is widely used by diverse research fields such as Monte Carlo simulations (YAMAKOV, 2016), drug design (MORETTI; SARTORI, 2016), big data analysis (REYES-ORTIZ; ONETO; ANGUITA, 2015) and neural networks (SCHUMAN et al., 2016).

According to Booth, Kim e Rajamanickam (2016), archiving parallel performance on chosen hardware architecture depends on factors such as scheduler overhead, data/task granularity, cache fitting, and data synchronization. There exist different abstraction level of parallelism strategies that can be combined (CHOUDHURY et al., 2015). Often, a systematic comparison between parallelism strategies is necessary to verify which one has better performance (BOOTH; KIM; RAJAMANICKAM, 2016). A comparison of two parallelism strategies in a Beowulf cluster is shown by Hulsey e Novikov (2016).

Data Parallelism strategy, as stated by Gordon, Thies e Amarasinghe (2006), is when one processing data slice does not have dependency with the next one. Thus, data is divided into several data slices and processing them equally by different processors. Task Parallelism objective is to spawn tasks across processors to speed up one scalable algorithm. Tasks can be spawned by a central task system or by a distributed task system, both adding processing overhead, with a distributed task system achieving less overhead (BOOTH; KIM; RAJAMANICKAM, 2016).

2.2 Delayed Transfer Entropy

A brief introduction to information theory topics is presented to help readers understand DTE itself. By no means it tries to cover the whole information theory field, but selected topics needed to understand DTE.

2.2.1 Information Theory concepts

According to [Pierce \(2012\)](#)¹, seminal paper ([SHANNON, 1948](#)) organized a general theory about communications, or as a more common term, information theory. The problem Shannon gave himself and solved was how to encode a given message so it can be sent the fastest way possible in a noisy channel with minimum error ([PIERCE, 2012](#)). Although people like Harry Nyquist and Ralph Hartley studied the area before, [Shannon \(1948\)](#) was responsible for elevating information theory as an accepted field of research.

Despite information theory origin from electrical communication studies, information theory is a mathematical theory in itself, therefore can be applied to diverse research fields ([PIERCE, 2012](#)).

As defined by ([SHANNON, 1948](#)), a communication system is composed of five parts [This author adapted some examples.]:

Information source Where the message to be transmitted is created. There are many types of messages: A sequence of letters in an email; an arbitrary function $f(t)$ in time; several functions in a television broadcast signal; a computer file; a multi-channel audio stream from a blue-ray;

Transmitter Transform the message to enable its transmission through a channel. For example, when sending an email using a laptop, the email is transformed in electromagnetic waves by wireless card transmitter; or transforming telephony voice sound pressure into an electrical current;

Channel Medium where the message is passed. For example: optical fibers, air frequency band, electrical wires.

Receiver Reconstruct the message from transmitted signal;

Destination Person or thing to receive the message.

To address how much information on average is necessary to encode a message, Shannon Entropy was proposed ([SHANNON, 1948](#)). Paraphrasing ([PIERCE, 2012](#)), "The

¹ The book ([PIERCE, 2012](#)) was chosen as a reference for basic information theory concepts as John R. Pierce was considered intellectual sparring partners to Shannon, and the book was reviewed by Shannon itself

entropy of a signal source in bits per symbol or per second gives the average number of binary digits per symbol or per second, necessary to encode the messages produced by the source", which is given in mathematical terms:

$$H(x) = -K \sum_{i=1}^n p_i \log p_i \quad (2.1)$$

where p_i is probability of symbol i^{th} in the message and K being a positive constant to conveniently choice unit of measure. Note that Pierce choose base two logarithm in his definition of Shannon Entropy (from now on, called just entropy), so the result is expressed in bits and is adopted for this study². From this point, entropy of a joint event is:

$$H(x, y) = - \sum_{i=1, j=1}^{n, m} p(i, j) \log_2 p(i, j) \quad (2.2)$$

where x and y are events with respectively n and m symbols and $p(i, j)$ the joint occurrence probability for events $x = i$ and $y = j$. Now supposing events x and y as not completely independent, we can assume that for each event x with value i , there exist a conditional probability $p_i(j)$ of event y being j , therefore conditional entropy is expressed as:

$$H_x(y) = - \sum_{i=1, j=1}^{n, m} p(i, j) \log_2 p_i(j) \quad (2.3)$$

where x and y are events with respectively n and m symbols; $p(i, j)$ the joint occurrence probability for events $x = i$ and $y = j$; and $p_i(j)$ the conditional probability of event $y = j$ for event $x = i$; By knowing the equivalence,

$$p_i(j) = \frac{p(i, j)}{\sum_{j=1}^m p(i, j)} \quad (2.4)$$

and with some mathematical operations, 2.3 can be expressed as:

$$H_x(y) = - \sum_{i=1, j=1}^{n, m} p(i, j) \log_2 \frac{p(i, j)}{p(j)} \quad (2.5)$$

By using 2.5, (COVER; THOMAS, 2012) shows a derived measurement called Mutual Information (MI), presented in equation 2.6. Intuitive Mutual information definition is how much x uncertainty can be reduced by knowing y and vice versa.

$$I(x; y) = \sum_{i=1, j=1}^{n, m} p(i, j) \log_2 \frac{p(i, j)}{p(i)p(j)} \quad (2.6)$$

² Other normally used logarithm bases are 10 and e . For base 2, unit "shannon" is also used as an alias for entropy bits

where x and y are events with respectively n and m symbols; $p(i, j)$ the joint occurrence probability for events $x = i$ and $y = j$; $p(i)$ the conditional probability of event $x = i$; and $p(j)$ the conditional probability of event $y = j$; Mutual information can also be expressed in terms of entropy:

$$I(x; y) = H(y) - H_x(y) \quad (2.7)$$

as MI is symmetric, x and y can be swapped with the same results.

2.2.2 Delayed Transfer Entropy

Transfer Entropy (TE) measurement shown in Equation 2.8 was introduced by (SCHREIBER, 2000) and is useful to measure information transfer between two time series. TE has an asymmetric nature, being possible to determine information transfer direction.

$$TE(x \rightarrow y) = \sum_{y_{n+1}, y_n, x_n} p(y_{n+1}, y_n^{(k)}, x_n^{(l)}) \log_2 \frac{p(y_{n+1} | y_n^{(k)}, x_n^{(l)})}{p(y_{n+1} | y_n^{(k)})} \quad (2.8)$$

where y_n and x_n denotes value of x and y at time n ; y_{n+1} the value of y at time $n + 1$; p is the probability of parenthesis content; l and k are the number of time slices used to calculate probability density function (PDF) using past values of x and y , respectively; According to (HLAVÁČKOVÁ-SCHINDLER et al., 2007), transfer entropy is equivalent to conditional MI:

$$TE(x \rightarrow y) = I(x^-, y^+ | y^-) \quad (2.9)$$

where $-$ superscription represents past events and $+$ represents current events. The delay between y^- and y^+ (respectively y_n and y_{n+1} on Equation 2.8) is calculated by a process named here as embedding.

Assuming $k = 1$ and $l = 1$ to simplify analysis (Also called as D1TE by (ITO et al., 2011)), TE algorithm is demanding regarding computational power (SHAO et al., 2015), with its computational complexity being $O(B^3)$, where B is the chosen number of bins in PDF.

An extension to D1TE proposed by (ITO et al., 2011) is delayed transfer entropy (DTE - Equation 2.10) , which is a D1TE with variable causal delay range. This way, a parameter d represents a variable time delay between y and x . DTE is a useful metric to determine where within d range, occurs the biggest transfer of information from X to Y .

$$DTE_{X \rightarrow Y}(d) = \sum_{y_{n+1}, y_n, x_{n-d}} p(y_{n+1}, y_n, x_{n-d}) \log_2 \frac{p(y_{n+1} | y_n, x_{n-d})}{p(y_{n+1} | y_n)} \quad (2.10)$$

2.2.3 Statistical Significance

TE has a non-negligible rate of false positives hence the importance of statistical testing. (VICENTE et al., 2011) affirmed that false rate happens because TE is zero when signals are fully independent and also when signals are identical. To avoid false positives, some methods were proposed to validate TE, such as null hypothesis tests against surrogate signals (SANTOS; MACIEL; NEWLAND, 2017) (SHARAEV; USHAKOV; VELICHKOVSKY, 2016). An general DTE example is shown on Figure 1.

Figure 1: Data analysis and significance levels. From input data $x(t)$, it is estimated a series of surrogate signals $X_n^S(t)$, and for each of surrogate signal is determined time delayed transfer entropy. The n repetitions indicate a significance level from Equation 2.11 against random probabilities with the same power spectra and amplitude distribution as real data. Assuming the number of surrogate n equals three as shown in this figure, DTE Threshold line displayed on biggest ensemble plot has a significance level of 75%.

The word surrogate stands for something that is used instead of something else. In the case of surrogate signals (DOLAN; SPANO, 2001), the synthetic data used is randomly generated, but it also presents some characteristics of the original signal that it is taking place. Even surrogate has the same power spectrum that the original data, both signals are uncorrelated. Different computational packages present algorithms to generate surrogate signals (LINDNER et al., 2011; MAGRI et al., 2009).

The Amplitude Adjusted Fourier Transform (AAFT) method explained by [Lucio, Valdés e Rodríguez \(2012\)](#) proposes rescaling the original data to a Gaussian distribution using fast Fourier transforms (FFT) phases randomization and inverse rescaling. This procedure introduces some bias, which [Lucio, Valdés e Rodríguez \(2012\)](#) explained a method to remove it by adjusting the spectrum from surrogates, named Iterative Amplitude Adjusted Fourier Transform (IAAFT) ([SCHREIBER; SCHMITZ, 1996](#)).

Surrogate signals generated by IAAFT algorithm ([SCHREIBER; SCHMITZ, 1996](#)) preserve the power density spectrum and probability density functions, but with the phase components randomly shuffled ([VENEMA; AMENT; SIMMER, 2006](#)). IAAFT algorithm is interesting for neurophysiological data, as causal association happens in phase synchronization ([YANG et al., 2013](#)).

[Santos, Maciel e Newland \(2017\)](#) generated surrogate signals applying IAAFT method and made a null hypothesis test that checked if original DTE peaks are higher than average DTE from surrogate signals, illustrated by [Figure 2](#). [Sharaev, Ushakov e Velichkovsky \(2016\)](#) null hypothesis tested if differences between TE from signals and TE from surrogates are a normal distribution with mean zero. [Mao e Shang \(2017\)](#) added synthetic white noise and uniformly distributed noise to data and repeated it fifty times for each noise type, varying white noise standard deviation and noise uniform noise level. [Mao e Shang \(2017\)](#) test showed that the TE result with added noise was between -5% and 5% for white noise and even smaller for uniform distributed noise. [Mao e Shang \(2017\)](#) concluded that adding noise to time series has little influence in TE result within a defined error range.

Figure 2: Adapted from [Santos, Maciel e Newland \(2017\)](#). First a pre processing is executed on signals to remove non-linear trends on data. Then 10 surrogates are generated for each signal. Finally, DTE from original signal is normalized against average DTE from surrogate signals

The importance of surrogate repetitions number is explicit derived from equation [Equation 2.11](#) presented is [Schreiber e Schmitz \(2000\)](#), where the amount of surrogate repetitions n is inverse proportional to the desired significance level α , and $k = 1$ for one

sided hypothesis test or $k = 2$ for two sided hypothesis test.

$$n = \frac{K}{\alpha} - 1 \tag{2.11}$$

3 MATERIAL AND METHODS

A brief description of methods is presented in this introduction, followed by a section for hardware setup, one for software setup, one describing DTE application and the remaining sections are for each study investigation step (From now on, we call each of them a "Part"). We opted for this scheme because although the Parts share hardware and software, each part has different methods.

The first study investigation was done to theoretically and empirically confront DTE algorithmic complexity against existing literature. Initially, empirical verification was done in an *ad hoc* manner, by manually changing parameters. Later, we came back to methodically measure complexity by varying input variables, inspired by [Coppa, Demetrescu e Finocchi \(2012\)](#). Thereafter we calculated our DTE algorithm theoretical complexity. This step was named **Part I - DTE time efficiency**.

Once we made sure that our implementation is correctly bounded by asymptotic big-O complexity, we started to optimize Transfer Entropy code as it was our hot path code according to **Part I - DTE time efficiency**. The study which led to performance improved is presented as **Part II - Transfer Entropy serial optimization**.

The next natural step was to make DTE application execute on the cluster automatically. Up to **Part II - Transfer Entropy serial optimization**, cluster utilization was done by manually partitioning the data across cluster nodes, logging in each node and executing DTE application. The simple act of starting an experiment was error prone and time consuming due to its *ad hoc* nature. DTE application clusterization didn't bring any performance improvement, but it made reproducibility easier to achieve and decreased chances of human error. Code improvements to allow cluster environment execution is referenced on this study as **Part III - DTE application cluster intrinsics**.

During some experiments execution using **Part III - DTE application cluster intrinsics** code, it was observed cluster sub-utilization. To overcome lower than expected cluster processing power, a study about different parallelism strategies was conducted and is shown in **Part IV - DTE application cluster optimization**

After finding a lack of good material for building an *ipyparallel* cluster we built a step-by-step cluster configuration guide. Furthermore, targeting open sourcing *lpslib* we did a code refactor on *lpslib* to improve easy of use, signal type extensibility and documentation. Both cluster and *lpslib* documentation is explained on **Part V - *lpslib* documentation**

3.1 Hardware Setup

Our hardware setup was composed of three computers used in software development and a computer cluster for exploring different types of parallelism. Laptop 1 ¹ has an Intel(R) Core(TM) i7-4500U CPU @ 1.80GHz, 2x4 GiB DDR3 1600MHz RAM memory and a Seagate ST1000LM024 HN-M 1 TB hard drive. Laptop 2 has an Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz, 1x16 GiB DDR4 2400MHz RAM memory and a LITEON CV3-8D128 128 GB solid state disk; And a Samsung Chromebook 3 with MrChromebox BIOS to enable a custom operating system.

Figure 3: Beowulf cluster used in the experiments. Lines between devices may represent multiple Ethernet cables for diagram cleanness purpose. Cluster nodes are prefixed with "lps" following by identification number. Detailed cluster configuration is informed in [Table 1](#).

Our Heterogeneous Beowulf cluster is composed of 10 nodes connected by Gigabit Ethernet show in [Figure 3](#). Full Beowulf cluster configuration is listed in [Table 1](#).

¹ Hardware information collected using the *lshw* tool.

Table 1: Cluster hardware configuration. RAM modules were list separately since some nodes have multiple memory modules to explore dual channel. Main storage describe storage media used during script execution, some nodes might have other unused storage media.

Node	Processor (cores)	RAM (speed)	Main Storage size (model)	Ethernet
host	i5-2500 CPU @ 3.30GHz	4 + 4 GiB (1333MHz)	2TB WDC WD20EARX-00P	Gigabit
lps01	i7-4770 CPU @ 3.40GHz (8)	8 + 8 GiB (1333MHz)	1TB ST1000DM003-1CH1	Gigabit
lps02	i7-3770 CPU @ 3.40GHz (8)	8 GiB (1333MHz)	60GB KINGSTON SV300S3	Gigabit
lps04	i7-4820K CPU @ 3.70GHz (8)	8 GiB (1333MHz)	2TB ST2000DM001-1CH1	Gigabit
lps05	i7-4820K CPU @ 3.70GHz (8)	8 GiB (1333MHz)	1863GiB ST2000DM001-1CH1	Gigabit
lps06	i7-4820K CPU @ 3.70GHz (8)	8 + 8 GiB (1333MHz)	60GB KINGSTON SV300S3	Gigabit
lps08	i7 950 CPU @ 3.07GHz (8)	4 + 4 + 4 GiB (1066MHz)	2TB ST32000542AS	Gigabit
lps09	i7-4790 CPU @ 3.60GHz (8)	8 + 8 GiB (1600MHz)	256GB SMART SSD SZ9STE	Gigabit
lps10	i7-4790 CPU @ 3.60GHz (8)	8 + 8 GiB (1600MHz)	256GB SMART SSD SZ9STE	Gigabit
lps11	i7-4790 CPU @ 3.60GHz (8)	8 + 8 GiB (1600MHz)	256GB SMART SSD SZ9STE	Gigabit
lps12	i7-4790 CPU @ 3.60GHz (8)	8 + 8 GiB (1600MHz)	256GB SMART SSD SZ9STE	Gigabit

3.2 Software setup

During development, it was given strong preference for Open Source software to address concerns about reproducibility as show by [BAKER \(2016\)](#). Using Open Source software allows inspecting the code to check for any errors, future-proof by enabling updates to run software in newer systems and more critical, considerably decreases cost and effort to repeat experiments.

This study had as a starting point a preexisting software library, named *lpslib* after *Laboratório de Processamento de Sinais*. *lpslib* was codified using Python ([ROSSUM; DRAKE, 2011](#)) programming language. We choose to keep the same language as Python has a good track record on scientific community (ref), offers increased development speed and performance with its optimized libraries.

Preexisting *lpslib* software library is referenced as CODE A. Each resulting code after each Part (explained in [chapter 3](#)) receives a name to make easier to reference on text. A detailed timeline on how the code evolved is show in [Figure 4](#). An overview from *lpslib* DTE application is presented in following [section 3.3](#)

The *lpslib* made use of several libraries, including: *numpy* ([WALT; COLBERT; VAROQUAUX, 2011](#)), responsible for optimized matrix operations; *scipy* ([JONES et al., 2001](#)–), with its for scientific routines; *matplotlib* ([HUNTER, 2007](#)) and *seaborn* to create plotting outupts; and IPython ([PÉREZ; GRANGER, 2007](#)) to enable cluster parallel processing.

About Python interpreter, we briefly investigated optimized *pypy* ([MAROWKA, 2018](#)) interpreter with our preexisting code, but we didn't find any major speed difference, as most running time was spent on native *numpy* methods. Due *pypy* similar performance for our use case and it's lack of modern Python (version 3.x +) support at the time, we opted to continue with *cpython* interpreter.

Figure 4: This graph show the resumed commit history. Note that although **Part I** was created temporally after **Part IV**, we did an prior *ad hoc* investigation. Its important to note that **Part I** measures only DTE performance in local mode.

The developed source code was stored in version control system (EGLEN et al., 2017) and good software engineering practices were adopted such as readability and modularization. According to (CHACON; STRAUB, 2014), main advantages to using the version control system are version control itself, peer collaboration, bisect old versions to look for bugs and keep commented history. Our chosen software was *git*, which has open source license and is widely used by academia and industry. Adding to *de facto git* standard argument, our option is backed by (RAM, 2013) discussion on how *git* can improve reproducibility and transparency in science

Additional supporting software employed during this research are: $\text{\LaTeX}$ as document preparing system, also used during this document creation; *TexStudio* as $\text{\LaTeX}$ text editor; *Spyder3* as Python integrated development environment; *Dia* to generate diagrams; *Inkscape* to edit vector files; *Gnuplot* to create plots; *coreutils* utilities to pre processes files; *ssh* for remote login; and *tmux* to manage terminal sessions.

Laptop 1 operating system is Arch Linux, Laptop 2 operating system is Ubuntu, and Chromebook operating system is an Ubuntu-based distribution named GalliumOS. Both development machines had their software updated during research without any implications since runtimes were collected within cluster nodes or with same software version. Our cluster software was kept constant during research and is listed in Table 2.

Table 2: Cluster software configuration. Updated at shows the date when each cluster node was last fully updated.

Node	Operating System (updated at)	numpy	IPython	pyfftw	Linux kernel
host	Fedora 24 Workstation (2016-08-17)	1.11.0	3.2.1	0.10.3.dev0+e827cb5	4.6.6-300.fc24.x86_64
lps01	Fedora 24 Server (2016-08-16)	1.11.0	3.2.1	0.10.3.dev0+e827cb5	4.6.6-300.fc24.x86_64
lps02	Fedora 24 Server (2016-08-16)	1.11.0	3.2.1	0.10.3.dev0+e827cb5	4.6.6-300.fc24.x86_64
lps04	Fedora 24 Server (2016-08-16)	1.11.0	3.2.1	0.10.3.dev0+e827cb5	4.6.6-300.fc24.x86_64
lps05	Fedora 24 Server (2016-08-16)	1.11.0	3.2.1	0.10.3.dev0+e827cb5	4.6.6-300.fc24.x86_64
lps06	Fedora 24 Server (2016-08-16)	1.11.0	3.2.1	0.10.3.dev0+e827cb5	4.6.6-300.fc24.x86_64
lps08	Fedora 24 Server (2016-08-16)	1.11.0	3.2.1	0.10.3.dev0+e827cb5	4.6.6-300.fc24.x86_64
lps09	Fedora 24 Workstation (2016-08-16)	1.11.0	3.2.1	0.10.3.dev0+e827cb5	4.6.6-300.fc24.x86_64
lps10	Fedora 24 Server (2016-08-16)	1.11.0	3.2.1	0.10.3.dev0+e827cb5	4.6.6-300.fc24.x86_64
lps11	Fedora 24 Server (2016-08-16)	1.11.0	3.2.1	0.10.3.dev0+e827cb5	4.6.6-300.fc24.x86_64
lps12	Fedora 24 Server (2016-08-16)	1.11.0	3.2.1	0.10.3.dev0+e827cb5	4.6.6-300.fc24.x86_64

3.3 DTE application

Bundled within *lpslib* library there is an example application with a complete method to use DTE. **DTE application** file is named *DTEexec.py* and it has a companion **DTE plot script** to plot results named *DTEAnalysis.py*. This section objective is give an overview on how **DTE application** logically works. This description applies for **CODE F** and its relation with old codes are explained by library *README.md* file found on section C.3, copied from *lpslib* root folder. For a detailed up-to date information on how to run **DTE application**, we recommend looking at *lpslib*'s *README.md* section "2 - Application example".

Figure 5: Component diagram showing **DTE application** (*DTEexec.py*) and **DTE plot script** (*DTEAnalysis.py*) relation to *lpslib* source files. *EmbeddingParams* method is responsible for finding Equation 2.9 delay between y^- and y^+ .

Command line to invoke **DTE application** is given by:

```
1 python3 DTEexec.py [-i <experiment file>] [--nsur <nsur>] [--lag <lag>]
2                   [--nbins <nbins>] [-r <signal reader name>]
3                   [-t <target folder>]
```

DTE application reads an experiment file listing all signal names that the user wants to calculate DTE. Each line of the experiment file contains '<signal name> <start sample> <end sample>', where <start sample> and <end sample> truncates into interested signal range; commented signals start with hash tag (#) and they are skipped; 0 for <start sample> and <end sample> means first and last sample respectively. An experiment file example is listed on [Listing 3.1](#).

Listing 3.1: Example of DTE application experiment file

```
1 ID-706_ssa 0 0
2 ID-774_ssa 20000 100000
3 ID-934_ssa 0 0
4 #ID-945_ssa 0 0
```

Experiment file from [Listing 3.1](#) contains 4 signal, with last signal being commented. The second signal is truncated between samples 20000 and 100000. First and third signals are totally used.

After parsing experiment file, DTE application reads the signal using one of the SignalReader classes passed as a parameter. SignalReader classes return the number of channels in the signal, sample rate, and signal data series.

With loaded signal data series, DTE application calculates embedding for each signal channel. Signal embedding is calculated by finding the target variable's past, which can be found in the first local minimum of auto-mutual information or first zero crossing auto-correlation measures ([KANTZ; SCHREIBER, 2004](#)).

Afterwards, DTE application will execute for each signal, a permutation between each channel. For each permutation, DTE is calculated to original channels with [Algorithm 1](#). This DTE result on the original signal is saved in Python's pickle serialization format with name <signal name>_<channel x>_<channel y>_REF.dte.

Algorithm 1 DTE algorithm

Ensure: signalX**Ensure:** signalY**Ensure:** tau {Embedding signalY lag}**Ensure:** nbins {number of pdf bins}

```

1: virtualLength  $\leftarrow$  length(signalX) - maxLag
2: for lag = 0 to maxLag do
3:   xd  $\leftarrow$  signalX[maxLag-lag : maxLag-lag + virtualLength]
4:   yds  $\leftarrow$  signalY[maxLag-tau : maxLag-tau + virtualLength]
5:   y  $\leftarrow$  signalY[maxLag : maxLag + virtualLength]
6:   hxdydsy  $\leftarrow$  histogramdd([xd, yds, y], nbins)
7:   pxdydsy  $\leftarrow$  hxdydsy / sum(hxdydsy)
8:   pxdyds  $\leftarrow$  sum(pxdydsy, axis=2)
9:   pydsy  $\leftarrow$  sum(pxdydsy, axis=0)
10:  pyds  $\leftarrow$  sum(pxdydsy, axis=(0,2))
11:  TE  $\leftarrow$  0
12:  for i = 0 to nbins do
13:    for j = 0 to nbins do
14:      for l = 0 to nbins do
15:        TE  $\leftarrow$  TE + pxdydsy[i, j, l] * log2(pxdydsy[i, j, l] * pyds[j] / (pydsy[j, l] * pxdyds[i, j]))
16:      end for
17:    end for
18:  end for
19: end for

```

Finally, DTE application generates *nsur* surrogate signals using Algorithm 2 for all second channel of each permutation tuple. For each surrogate, DTE is calculated and saved together as Python's pickle serialization format file, named *<signal name>_<channel x>_<channel y>_SUR.dte*

Algorithm 2 IAAFT

```

1: original_frequency_bins  $\leftarrow$  FFT(original_signal)
2: original_frequency_amplitude  $\leftarrow$  abs(original_frequency_bins)
3: original_frequency_phase  $\leftarrow$  phase(original_frequency_bins)
4: permuted_signal  $\leftarrow$  random_permutation(original_signal)
5: permuted_frequency_bins  $\leftarrow$  FFT(permuted_signal)
6: permuted_frequency_amplitude  $\leftarrow$  abs(permuted_frequency_bins)
7: while num_step < max_steps do
8:   permuted_frequency_phase  $\leftarrow$  phase(permuted_frequency_bins)
9:   new_frequency_bins  $\leftarrow$  original_frequency_amplitude*exp(j*permuted_frequency_phase)

10:  new_signal  $\leftarrow$  IFFT(new_frequency_bins)
11:  permuted_signal  $\leftarrow$  swap_vaues(original_singal, new_signal)
12:  permuted_frequency_bins  $\leftarrow$  FFT(permuted_signal)
13:  permuted_frequency_amplitude  $\leftarrow$  abs(permuted_frequency_bins)
14:  current_error  $\leftarrow$  error(permuted_frequency_amplitude,original_frequency_amplitude))
15:  if num_step = 0 then
16:    min_error = current_error
17:  end if
18:  if current_error < min_error then
19:    min_error = current_error
20:    best_result = real(new_signal)
21:  end if
22:  if num_step > 0 and abs(last_error-current_error) < stop_tolerance then
23:    return best_result
24:  end if
25: end while

```

We use **DTE plot script** generate plots in order to visualize results:

```
python3 DTEAnalysis.py [-i <experiment file>] [-t <target folder>]
```

For each signal in experiment file, **DTE plot script** reads DTE on original signal and DTE from *nsur* surrogates from each permutation an plots into a matrix-like layout. An **DTE plot script** example output is presented on [Figure 6](#).

Figure 6: **DTE plot script** matrix-like layout output example. Output file is in a vector format (Portable Document Format) with clear zoom visualization on electronic viewers. On bottom there is an artificially zoomed plot from permutation channel name 200 to channel name 27. On blue is the original signal DTE result, and other are DTE from surrogate signals, in this case, $nsur = 3$

3.4 Part I - DTE time efficiency

Before starting optimizing DTE, we did an informal verification of our implemented DTE algorithmic complexity by manually varying input parameters and observing total running time. In order to change DTE input we used **CODE A** to changed maximum delay, the number of PDF discretization bins, and signal duration. Varying signal duration was done by swapping experiment file with different signal lengths.

Command used to informally verify our DTE implementation:

```
python3 execDTE.py --lag <LAG> --nbins <NBINS>
```

Where LAG represented maximum delay between DTE input and output signals (max value of parameter d of [Equation 2.10](#)); NBINS the number of PDF discretization bins.

After **Part IV - DTE application cluster optimization** was done, we returned to verify our implemented DTE time efficiency against existing literature formally. We did a theoretical analysis by looking at the transcribed [Algorithm 1](#). Time efficiency represents CPU utilization time according to algorithm input.

We employed the following method to find theoretical algorithm time efficiency ([LEVITIN, 2012](#)):

- 1 Choose input parameter(s)
- 2 Identify algorithm inner-most basic operation
- 3 Verify if inner-most operation depends on a condition. If so, worst, average and best cases should be investigated individually
- 4 Express in terms of sums the number of basic operations
- 5 Use manipulations to establish the asymptotic order of growth

Where basic operation corresponds to the operation taking most CPU time, in general, the innermost loop. A complete description of this method, including examples and special cases, is presented by ([LEVITIN, 2012](#), Chapter 2, Section 3).

As discussed in following [section 4.1](#), DTE does not have special conditions on the inner-most basic operation; therefore DTE has a deterministic running time. We choose to not detail methodologies differences between worst, average and best case scenarios because they would be equal for DTE.

In addition to theoretical verification, we did an empirical study to increase evidence of our DTE implementation correctness. We use a methodology to find asymptotic inefficiencies by profiling implemented code, inspired by [Coppa, Demetrescu e Finocchi \(2012\)](#). The idea is to profile the DTE application with varying input size to estimate the growth rate. With resulting tuples of input size vs. profile, we are able to plot runtime growth and therefore, estimate asymptotic rate by curve fitting. The algorithm for input based profiling is shown in [3](#). It is important to note that input based profiling is an evidence against implementation hidden inefficiencies, it does not give algorithmic complexity.

Algorithm 3 Input based profiling

```

1: result  $\leftarrow$  []
2: for inputSize = 0 to maxInputSize do
3:   input  $\leftarrow$  generateInput(inputSize)
4:   startProfiling()
5:   func(input)
6:   profilinResult  $\leftarrow$  stopProfiling()
7:   result  $\leftarrow$  result + [(inputSize, profile)]
8: end for

```

We branched **CODE E** from *master* branch, and transformed *ExecDTE.py* into *benchDTE.py* with removal of *ipyparallel* code and adding profiling capabilities by wrapping DTE with built-in Python library *cProfile* *runcall* method.

Looking at input based profiling Algorithm 3, added *cProfile* wrapper was responsible for line 4 *startProfiling()* and line 6 *profilinResult* $\leftarrow$ *stopProfiling()*; while DTE was line 5 *func(input)*; maximum delay, number of PDF discretization bins and signal duration were chosen as input for input based profiling. We did not disabled power save features neither Intel Turbo Boost from our Laptop 2, hence small deviations are expected.

To measure growth rate according to input, we executed command:

```
1 python3 benchDTE.py
```

Input based profiling results were fitted against equation Equation 3.1 using *scipy curve_fit* method and displayed on plot output. *curve_fit* is backed by Trust Region Reflective method presented by Branch, Coleman e Li (1999).

$$f(x) = a + (x * b)^c \quad (3.1)$$

Results can be plotted by command:

```
1 python3 benchPlot.py
```

The informal verification was executed on the Laptop 1, and the input based profiling was executed on Laptop 2.

3.5 Part II - Transfer Entropy serial optimization

After getting comfortable with DTE software (**CODE A**), the first logical step was to increase serial performance. Code profile on **CODE A1** was done to determine code paths where most computing time was spent. A built-in *cpython* module named *cProfile* was used:

```
1 python -m cProfile -o output.txt ExecDTE.py
```

Profile result was analyzed with open source program KCacheGrind, after processing *cProfile* output with *pyprof2calltree* Python package:

```
1 pyprof2calltree -i output.txt -o callgrind.filename.prof
```

With annotated execution time for each function, we were able to optimize slower code paths effectively. After optimizing DTE, we added a comparison between DTE and its now optimized version on **CODE A2** to measure performance gain.

CODE A2 executed 10 optimized DTE and 10 non-optimized DTE. Running times were manually parsed from the output and gathered into a simple plotting script, on the annex. Each optimized and non-optimized DTE were repeated 10 times, showed on resulting box plot.

After some cleanup we achieved **CODE B**, on *master* branch.

This part was executed on the Laptop 1.

3.6 Part III - DTE application cluster intrinsics

Inherited DTE software version (**CODE A**) was executed by splitting experiment list file into smaller experiment list files and creating an instance for each of these smaller files. To run each instance, first we would remote login using ssh, open *tmux* and execute each instance manually.

Although we were using bash scripts to automate execution, it was an error-prone and tedious experience. To alleviate this, *ipyparallel* was configured and **CODE B** refactored to use parallel intrinsics to automatically execute task across cluster nodes, named here as **CODE C**.

The first parallelization was tested using *ipyparallel* as a local machine cluster (e.g. every processor core from local machine is a cluster node), then *ipyparallel* was configured to assemble a cluster with all nodes.

The configuration of *ipyparallel* cluster included on each node and host: hard drive formatting, installing an operating system, programs, libraries and, settings configuring. This step although not scientific *per se* was further documented to share technical knowledge.

To check performance we used a dataset composed of 35 neurophysiological signals each with four simultaneous captured channels. The average number of signal samples is about 1 million samples with a standard deviation of about 500 thousand samples.

This part was executed on cluster.

3.7 Part IV - DTE application cluster optimization

During execution of experiments using **CODE C**, we launched *htop* (Bartosz Fenski et al., 2016) process monitor and verified that load average was below number of core, meaning a sub utilization according to *proc* manual (Linux Developers, 2016).

Further investigation by adding extensive logging facilities to **CODE C** confirmed that the low load average was caused by network communication bottleneck. The code was refactored to use task-based parallelism (**CODE D**), aiming to mitigate communication bottleneck. The idea is to load signal data from local storage instead of network transfer from the host node.

Both **CODE C** and **CODE D** were executed with a different number of surrogates (1, 5, 10 and 20) to compare performance between them, except 20 surrogates for **CODE A** due excessively long execution time (estimated in more than 6000 minutes by extrapolating results from **CODE C** with a smaller number of surrogates). To give some perspective on our experiment data size, for Data and Task Parallelism experiments involving ten surrogates, the total number of calculated TEs is about 11,642,400 (35 signals x 12 channel pairs – 2-permutations of 4 channels – x 2520 TE/permutation – refers to variation of DTE d delay parameter – x (1 original data + 10 surrogate)).

Resulting logs were processed to calculate the duration for each execution. Linear least square method was used to fit a line for Data and Task Parallelism duration. By supposing surrogate creation is insignificantly in comparison with DTE duration, each line slope represents Minutes/surrogate. Finally, the speedup was calculated using line slopes to measure performance gain from Task Parallelism over Data Parallelism.

This part was executed on the cluster using the same dataset from **Part III - DTE application cluster intrinsics**.

3.8 Part V - *lpslib* documentation

Documentation was written using a markup format named Markdown (OVADIA, 2014). Markdown is simple to write and can be transformed into a variety of document formats using tools. Due to its plain text nature, it gained popularity within scientific and programming communities.

We created a *"README.md"* file in the root directory to document *lpslib* behavior and usage. A detailed manual was created explaining cluster intallation from formatted machines to final *ipyparallel* cluster with *Jupyter* notebook support (PERKEL, 2018). A guide on how to use *ipyparallel* to parallelize code was also created. Finally, explanatory code comments were written through *lpslib* source code.

This part was written on development machines.

4 RESULTS AND DISCUSSION

Results are divided according to each Part listed in [chapter 3](#). We grouped both results and discussion under each Part to make easier to follow.

4.1 Part I - DTE time efficiency

First informal investigation showed that the number of bins had the most influence on runtime. The informal investigation corroborated with [Shao et al. \(2015\)](#) that showed that the complexity of TE is $O(nbins^3)$, where *nbins* is the chosen number of bins in PDF. Since DTE is done by successively applying TE across maximum delay, it made sense.

On our DTE algorithmic time efficiency retake, the theoretical result was achieved following proposed method on [section 3.4](#) on **CODE D**. As input parameter, we choose maximum delay, number of PDF discretization bins, and signal duration. DTE inner most basic operation is equivalent of line 15 from Algorithm 1. Expressing the number of basic operations in terms of sum:

$$C(maxLag, nbins, singalDuration) = \sum_{lag=1}^{maxLag} \sum_{i=1}^{nbins} \sum_{j=1}^{nbins} \sum_{k=1}^{nbins} 1 = maxLag * nbins^3 \quad (4.1)$$

Where $C(maxLag, nbins, singalDuration)$ represents number of basic operations in terms of maximum delay, number of PDF discretization bins, and signal duration. Note that although we choose *signalDuration* as input parameter, it does not affect the number of basic operations, but only PDF estimation method (equivalent of line 6 from Algorithm 1), which in our case, is logarithmic time for each sample according to *histogramdd* source code ¹.

Our DTE asymptotic order of growth according to [Equation 4.1](#) is $\mathcal{O}(maxLag * nbins^3)$, which is literally TE time efficiency ($\mathcal{O}(nbins^3)$, ([SHAO et al., 2015](#))) applied across *maxLag* time interval.

Empirical study using input based profiling started from **CODE D** by creating a new brach called *profile*; final code is named **CODE E**. We ran once for each input as this study is qualitative, with small variations not affecting asymptotic results. Figures [Figure 7](#), [Figure 8](#) and [Figure 10](#) shown the behavior of each input by varying its size.

[Figure 7](#) behavior is according to expected result of $\mathcal{O}(maxLag)$, since other inputs are fixed.

¹ As of *numpy* version 1.15, *histogramdd* method uses *searchsorted*, which is a binary search with a known time efficiency of $\mathcal{O}(\log nbins)$ for each sample

Figure 7: Input based profiling on DTE algorithm by varying maximum delay (lag) input

Figure 8 behavior is according to expected result of $\mathcal{O}(nbins^3)$, since other inputs are fixed. An idea on numpy broadcast performance gain, can be shown on input based profiling with **CODE B** in Figure 9. In innermost DTE multiplication, we use **numpy** broadcast only on one of $nbins$ dimension, which explain the result. It is important to remember that *numpy* broadcast optimization by no means decreased algorithmic complexity.

Figure 8: Input based profiling on DTE algorithm by varying number of PDF discretization bins (*nbins*) input

Figure 9: Input based profiling on DTE algorithm by varying number of PDF discretization bins (*nbins*) input for optimized **CODE B**

Figure 10 result from the signal duration based input is especially interesting because it shows that by fixing the number of PDF discretization bins and maximum delay, the time efficiency depends on *histogramdd* algorithm. It confirms our expected theoretical result ($\mathcal{O}(\text{signalDuration})$) because the innermost DTE multiplication is constant when we only vary signal duration, being line 6 from Algorithm 1 the replacing basic operation.

Figure 10: Input based profiling on DTE algorithm by varying signal duration (*signalDuration*) input

4.2 Part II - Transfer Entropy serial optimization

After executing profiler shown in Figure 11, signal binning (*histogramdd* method) and DTE calculation itself were spotted as opportunities to optimize. Signal binning is responsible for estimating PDF. DTE calculation involves looping through x^- , y^- and y^+ .

All necessary PDFs to calculate Equation 2.10 $p(y^+, y^-, x^-)$, $p(y^+|y^-, x^-)$, $p(y^+|y^-)$ were estimated by signal binning². CODE A calculated each PDF individually. Our approach was to calculate $p(y^+, y^-, x^-)$ and derive from it $p(y^+|y^-, x^-)$ and $p(y^+|y^-)$, avoiding two whole signal iterations.

² Other PDFs estimation methods are discussed by Lee et al. (2012)

Figure 11: Code profile result on CODE B DTE algorithm.

To speed up DTE calculation, several changes were tested, but the best result was achieved using *numpy* broadcast. Numpy broadcast is used to vectorizing matrix operations and execute them within a high-performance loop implemented in c.

The performance gain is shown in Figure 12 with calculated speedup presented in Equation 4.2.

Figure 12: DTE inner loop execution time for both CODE A and CODE B (optimized DTE). Data measured with ten surrogate signals with 686053 samples each.

$$Speedup = \frac{t_{CODE\ A}}{t_{CODE\ B}} = \frac{54.852}{19.947} = 2.75 \quad (4.2)$$

4.3 Part III - DTE application cluster intrinsics

To enable automatic parallelization, *ipyparallel* cluster was configured according to manual and code was modified. Asymmetric login authentication was implemented from the host machine to cluster nodes. *ipyparallel* configuration file is found in *lpslib doc* folder.

After several modifications, code was changed from CODE B displayed in Figure 13 to CODE C in Figure 14. Software modifications consisted mainly of transforming experiment loop with *ipyparallel* parallelization directives.

Figure 13: CODE B Serial algorithm flowchart. One instance of this program was executed for each slice of an experiment list. Objects of this study are highlighted in blue.

Figure 14: CODE C Data parallelism algorithm flowchart. Note that each dashed line means data transfer to every cluster node and every dotted line means data gathering and synchronization to host node. Objects of this study are highlighted in blue.

Achieved results indeed simplified experiment execution, allowing a faster iteration and decreasing error-prone manual steps.

4.4 Part IV - DTE application cluster optimization

After several modifications, code was changed from CODE C displayed in Figure 14 to CODE D in Figure 15. Software modifications consisted mainly of moving experiment execution loop to cluster nodes using *ipyparallel* parallelization directives.

Figure 15: CODE D Task parallelism algorithm flowchart. Note that each dashed line means data transfer to every cluster node and every dotted line means data gathering and synchronization to host node. Objects of this study are highlighted in blue.

Execution logs gave the total execution time per number of surrogates as shown in Figure 16 by the point marks, and the presented dashed lines are fitted by linear least square method. Line slopes for each parallelism strategy are 280.385 minutes/surrogate (Data Parallelism) and 65.257 minutes/surrogate (Task Parallelism). Therefore, speedup can be determined as shown in Equation 4.3.

Figure 16: Data and Task Parallelism total execution time per number of surrogate signals. Point marks show numerical results in minutes. Lines show data fitted by linear square method

$$Speedup = \frac{t_{Data \ Parallelism}}{t_{Task \ Parallelism}} = \frac{280.385}{65.257} = 4.297 \quad (4.3)$$

The achieved speedup, ~ 4.3 , shows that Task Parallelism is significantly faster than Data Parallelism. After analyzing logs, positive speedup can be explained by three main factors, and it is negatively impacted by another.

First speedup explanation is data locality since data is stored on local disk in Task Parallelism versus being transferred by the network in Data Parallelism. Also, former has to transfer channel data for every surrogate, while later, locally read signal data only once for each two-channel permutation surrogates.

Second factor is sub-optimum node utilization caused by cluster heterogeneity illustrated in Figure 17. This happens in Data Parallelism because data is equally divided across computing nodes with different performance, causing faster nodes, which finished data processing, to wait for slower nodes.

The third factor is caused by the fact of Data Parallelism surrogate datasets are generated only by host node, forcing all computing nodes to wait for surrogate dataset generation. Task Parallelism does not suffer from the same problem, as while one computing nodes generate one surrogate dataset, it does not block another computing node.

Figure 17: DTE Task Parallelism performance comparison between different computing nodes to highlight cluster heterogeneity. In Task Parallelism, each experiment spawned $number\ of\ channels * (number\ of\ channels - 1)$ DTE tasks of the same size across computing nodes, their execution times were used to calculate speedup of Y-axis computing node over X axis computing node and finally was made an average of calculated speedups for each cell. Execution log used to generate this plot was from 20 surrogates. Values are given in relative speedup

The asynchronous nature of tasks are negatively affecting Task Parallelism speedup when task pool is exhausted, some computing nodes are left without any task, reducing the speedup enhancement.

About node performance difference, [Figure 17](#) suggests that it may be caused by different random-access memory (RAM) sizes across cluster nodes, since slowest nodes (*lps02*, *lps04*, *lps05* and *lps08*) have smallest RAM amount (8 GiB, 8 GiB, 8 GiB and 12 GiB respectively) as listed in [Table 1](#) in Appendix.

An analogy can be made between MapReduce and presented parallelism strategies. In [Figures 14](#) and [15](#), dashed and dotted arrows would correspond to respectively

map and reduce operations. Keeping the same analogy, this study would be about the balance between communication and computation to optimize MapReduce DTE runtime performance.

4.5 Part V - *lpslib* documentation

Documentation was transformed from Markdown format into Portable Document Format (PDF) using *markdown-pdf* tool, in order to annex to this study. Command line to build PDF is show on [Listing 4.1](#)

Listing 4.1: Build PDF documentation

```
1 markdown-pdf <markdown input file>
```


5 CONCLUSION

DTE is a probabilistic non-linear measure to infer causality within a time delay between time-series. However, DTE algorithm demands high processing power, requiring approaches to overcome such limitation. A distributed processing approach was presented to speedup DTE computation using parallel programming over a heterogeneous low-cost computer cluster.

We started by validating our DTE implementation time efficiency against existing literature. Afterward, we optimized DTE running time with software profiling help. Then we modified **DTE program** to enable execution on cluster. Data and Task Parallelism strategies were compared to optimize software execution time. Finally we documented *lpslib*, **DTE program** and how to create an *ipyparallel* cluster with *Jupyter* support.

In **Part I - DTE time efficiency** we verified that our DTE implementation is according to scientific literature regarding algorithm complexity, both theoretically and empirically. Theoretical DTE time efficiency was achieved by finding the algorithm asymptotic growth behavior and validated against a TE literature extrapolation. Finally, we employed a methodology ([COPPA; DEMETRESCU; FINOCCHI, 2012](#)) to discover hidden asymptotic inefficiencies in order to corroborate with our DTE implementation correctness.

In **Part II - Transfer Entropy serial optimization** we profiled our DTE code and found that binning the signals create PDF and innermost multiplication was responsible for most of the running time. After some iterations of optimizing and profiling, we found that we could do binning once and, for innermost DTE multiplication, we could use *numpy* broadcasting ([WALT; COLBERT; VAROQUAUX, 2011](#)).

In **Part III - DTE application cluster intrinsics** we contributed by exploring low-cost Beowulf heterogeneous computer cluster as a new alternative to existent FPGA TE ([SHAO et al., 2015](#)) or GPU DTE ([WOLLSTADT et al., 2014](#)) implementations. The low-cost nature of Beowulf computer clusters and its simple setup enable using existing computers from research laboratories or universities, helping mitigate DTE performance issues without the acquisition of expensive hardware such as FPGAs or GPU cards. This is especially attractive to places where research funding lacks enough resources or where DTE usage is infrequently to justify any hardware purchases.

In **Part IV - DTE application cluster optimization** we used Task Parallelism strategy to increase DTE algorithm performance in a heterogeneous cluster was shown as a faster alternative in comparison to Data Parallelism. Acknowledging big data analysis importance, it is a significant result, since it will enable causal inference for bigger previous

inapplicable datasets or with better causality statistical evidence.

In **Part V - *lpslib* documentation** we documented *lpslib* library, including comments on source code. We also made a detailed manual on how to build an *ipyparallel* cluster with *Jupyter* notebook support. Although *ipyparallel* cluster guide is a technical activity *per se*, software infrastructure is beneficial for modern research as shown in recent Nature publication about *Jupyter* notebook (PERKEL, 2018).

Studying DTE applied to Big Data demands high processing power, for example, to increase confidence from 95.24% (n=20) to 99.9%(n=999), Task Parallelism run time is increased from 0.9 days to estimated 45.27 days using our setup and data from **Part IV - DTE application cluster optimization**. Although a long runtime, it is a notorious improvement from about half a year from extrapolated Data Parallelism runtime. This highlights the importance of hardware performance to increase statistical confidence and gives strong support to keep researching speedup methods for DTE.

Having verified Task Parallelism as a better approach to DTE in a heterogeneous Beowulf cluster, it remains open how the number of computing nodes affects performance. Thus, future research should investigate how scalable Task Parallelism is after an increased number of computing nodes observing Amdahl's law (HENNESSY; PATTERSON, 2011). Along the same lines, performance, scalability and cost analysis of renting Cloud Computing nodes to build a cluster on demand to explore causality in big data using DTE is needed.

An open question unique to our cluster configuration is if RAM amount correlates with performance in computing nodes when executing Task Parallelism as suggested by our results from Figure 17. Moreover, different parallelism strategies can be tested on a case by case aiming to speed up processing of the ever increasing data size.

Using open source software helped towards the goal to achieve full reproducibility of this research, a common concern across every science field (BAKER, 2016; Nature Editorial, 2016). In particular, *git* software improved source code trackability both in development and while presenting results.

From a scientific productivity perspective, this research yielded significant results illustrated by accepted **Conference on Complexity, Future Information Systems and Risk (COMPLEXIS), 2017** paper and published **Congresso Brasileiro de Automática (CBA), 2018** paper. Additionally, improvements made on *lpslib* library was employed by Santos, Maciel e Newland (2017), showing an indirect impact of the present study and highlighting the benefits of open source libraries.

BIBLIOGRAPHY

AMEUR, M. S. B.; SAKLY, A. Fpga based hardware implementation of bat algorithm. **Applied Soft Computing**, Elsevier, v. 58, p. 378–387, 2017.

BAKER, M. Is there a reproducibility crisis? **Nature**, v. 533, p. 452–454, 2016. ISSN 0028-0836.

BAREINBOIM, E.; PEARL, J. Causal inference and the data-fusion problem. **Proceedings of the National Academy of Sciences**, National Acad Sciences, v. 113, n. 27, p. 7345–7352, 2016.

Bartosz Fenski et al. **htop(1) Linux User's Manual**. [S.l.], 2016.

BAZOW, D.; HEINZ, U.; STRICKLAND, M. Massively parallel simulations of relativistic fluid dynamics on graphics processing units with cuda. **Computer Physics Communications**, Elsevier, 2017.

BERGER, E. et al. Experience-based torque estimation for an industrial robot. In: **2016 IEEE International Conference on Robotics and Automation (ICRA)**. [S.l.: s.n.], 2016. p. 144–149.

BESSANI, M. et al. Impact of operators' performance in the reliability of cyber-physical power distribution systems. **IET Generation, Transmission & Distribution**, IET, v. 10, p. 2640–2646, 2016.

BOOTH, J. D.; KIM, K.; RAJAMANICKAM, S. A comparison of high-level programming choices for incomplete sparse factorization across different architectures. In: **IEEE. Parallel and Distributed Processing Symposium Workshops, 2016 IEEE International**. [S.l.], 2016. p. 397–406.

BRANCH, M. A.; COLEMAN, T. F.; LI, Y. A subspace, interior, and conjugate gradient method for large-scale bound-constrained minimization problems. **SIAM Journal on Scientific Computing**, SIAM, v. 21, n. 1, p. 1–23, 1999.

CAMILLO, M. H. et al. Combining exhaustive search and multi-objective evolutionary algorithm for service restoration in large-scale distribution systems. **Electric Power Systems Research**, Elsevier, v. 134, p. 1–8, 2016.

CAO, G.; ZHANG, Q.; LI, Q. Causal relationship between the global foreign exchange market based on complex networks and entropy theory. **Chaos, Solitons & Fractals**, v. 99, p. 36 – 44, 2017. ISSN 0960-0779. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0960077917300991>.

CHACON, S.; STRAUB, B. **Pro Git**. 2nd. ed. Berkely, CA, USA: Apress, 2014. ISBN 1484200772, 9781484200773.

CHANG, V. Towards data analysis for weather cloud computing. **Knowledge-Based Systems**, Elsevier, v. 127, p. 29–45, 2017.

CHEN, C. P.; ZHANG, C.-Y. Data-intensive applications, challenges, techniques and technologies: A survey on big data. **Information Sciences**, Elsevier, v. 275, p. 314–347, 2014.

CHENG, D. et al. Improving performance of heterogeneous mapreduce clusters with adaptive task tuning. **IEEE Transactions on Parallel and Distributed Systems**, IEEE, v. 28, n. 3, p. 774–786, 2017.

CHOUDHURY, O. et al. Balancing thread-level and task-level parallelism for data-intensive workloads on clusters and clouds. In: IEEE. **2015 IEEE International Conference on Cluster Computing**. [S.l.], 2015. p. 390–393.

COOK, C. E. et al. The european bioinformatics institute in 2016: data growth and integration. **Nucleic acids research**, Oxford University Press, v. 44, n. D1, p. D20–D26, 2015.

COPPA, E.; DEMETRESCU, C.; FINOCCHI, I. Input-sensitive profiling. **SIGPLAN Not.**, ACM, New York, NY, USA, v. 47, n. 6, p. 89–98, jun. 2012. ISSN 0362-1340. Disponível em: <http://doi.acm.org/10.1145/2345156.2254076>.

COVER, T. M.; THOMAS, J. A. **Elements of information theory**. [S.l.]: John Wiley & Sons, 2012.

DEAN, J.; GHEMAWAT, S. Mapreduce: simplified data processing on large clusters. **Communications of the ACM**, ACM, v. 51, n. 1, p. 107–113, 2008.

DIEBOLD, F. X. A personal perspective on the origin (s) and development of 'big data': The phenomenon, the term, and the discipline, second version. 2012.

DOLAN, K. T.; SPANO, M. L. Surrogate for nonlinear time series analysis. **Physical Review E**, APS, v. 64, n. 4, p. 046128, 2001.

EGLIN, S. J. et al. Toward standard practices for sharing computer code and programs in neuroscience. **Nature neuroscience**, Nature Publishing Group, v. 20, n. 6, p. 770, 2017.

ENDO, W. et al. Delayed mutual information infers patterns of synaptic connectivity in a proprioceptive neural network. **Journal of Computational Neuroscience**, v. 38, n. 2, p. 427–438, 2015. ISSN 1573-6873. Disponível em: <http://dx.doi.org/10.1007/s10827-015-0548-6>.

GAILLARD, M. **CERN Data Centre passes the 200-petabyte milestone** | CERN. 2017. Disponível em: <https://home.cern/about/updates/2017/07/cern-data-centre-passes-200-petabyte-milestone>.

GANDOMI, A.; HAIDER, M. Beyond the hype: Big data concepts, methods, and analytics. **International Journal of Information Management**, Elsevier, v. 35, n. 2, p. 137–144, 2015.

GORDON, M. I.; THIES, W.; AMARASINGHE, S. Exploiting coarse-grained task, data, and pipeline parallelism in stream programs. **SIGARCH Comput. Archit. News**, ACM, New York, NY, USA, v. 34, n. 5, p. 151–162, out. 2006. ISSN 0163-5964.

GOU, J. et al. A novel improved particle swarm optimization algorithm based on individual difference evolution. **Applied Soft Computing**, Elsevier, v. 57, p. 468–481, 2017.

HARTICH, D.; BARATO, A. C.; SEIFERT, U. Sensory capacity: An information theoretical measure of the performance of a sensor. **Phys. Rev. E**, American Physical Society, v. 93, p. 022116, Feb 2016. Disponível em: <https://link.aps.org/doi/10.1103/PhysRevE.93.022116>.

HARUNA, T.; FUJIKI, Y. Hodge decomposition of information flow on small-world networks. **Frontiers in Neural Circuits**, Frontiers Media SA, v. 10, 2016.

HASHEM, I. A. T. et al. The rise of “big data” on cloud computing: Review and open research issues. **Information Systems**, Elsevier, v. 47, p. 98–115, 2015.

HENNESSY, J. L.; PATTERSON, D. A. **Computer architecture: a quantitative approach**. [S.l.]: Elsevier, 2011.

HIRATA, Y. et al. Detecting causality by combined use of multiple methods: Climate and brain examples. **PLoS ONE**, Public Library of Science, v. 11, n. 7, p. 1–15, 07 2016. Disponível em: <http://dx.doi.org/10.1371/journal.pone.0158572>.

HLAVÁČKOVÁ-SCHINDLER, K. et al. Causality detection based on information-theoretic approaches in time series analysis. **Physics Reports**, Elsevier, v. 441, n. 1, p. 1–46, 2007.

HU, H. et al. Toward scalable systems for big data analytics: A technology tutorial. **IEEE access**, IEEE, v. 2, p. 652–687, 2014.

HULSEY, S.; NOVIKOV, I. Comparison of two methods of parallelizing geant4 on beowulf computer cluster. **Bulletin of the American Physical Society**, APS, v. 61, 2016.

HUNTER, J. D. Matplotlib: A 2d graphics environment. **Computing In Science & Engineering**, IEEE COMPUTER SOC, v. 9, n. 3, p. 90–95, 2007.

ITO, S. et al. Extending transfer entropy improves identification of effective connectivity in a spiking cortical network model. **PLoS ONE**, v. 6, n. 11, 2011. ISSN 19326203.

JONES, E. et al. **SciPy: Open source scientific tools for Python**. 2001–. [Online; accessed 2016-06-23]. Disponível em: <http://www.scipy.org/>.

KANTZ, H.; SCHREIBER, T. **Nonlinear time series analysis**. [S.l.]: Cambridge university press, 2004. v. 7.

KAPP, M. N.; SABOURIN, R.; MAUPIN, P. A dynamic model selection strategy for support vector machine classifiers. **Applied Soft Computing**, Elsevier, v. 12, n. 8, p. 2550–2565, 2012.

KIRST, C.; TIMME, M.; BATTAGLIA, D. Dynamic information routing in complex networks. **Nature communications**, Nature Publishing Group, v. 7, 2016.

KITCHIN, R. Big data, new epistemologies and paradigm shifts. **Big Data & Society**, SAGE Publications Sage UK: London, England, v. 1, n. 1, p. 2053951714528481, 2014.

LEE, J. et al. Transfer entropy estimation and directional coupling change detection in biomedical time series. **BioMedical Engineering OnLine**, v. 11, n. 1, p. 19, Apr 2012. ISSN 1475-925X. Disponível em: <<https://doi.org/10.1186/1475-925X-11-19>>.

LEVITIN, A. **Introduction to the design & analysis of algorithms**. [S.l.]: Boston: Pearson,, 2012.

LI, G.; QIN, S. J.; YUAN, T. Data-driven root cause diagnosis of faults in process industries. **Chemometrics and Intelligent Laboratory Systems**, v. 159, p. 1 – 11, 2016. ISSN 0169-7439.

LIMA, D. R. de et al. Modeling of an insect proprioceptor system based on different neuron response times. In: . [S.l.: s.n.], 2016.

LINDNER, M. et al. Trentool: A matlab open source toolbox to analyse information flow in time series data with transfer entropy. **BMC neuroscience**, BioMed Central, v. 12, n. 1, p. 119, 2011.

Linux Developers. **proc(5) Linux User's Manual**. [S.l.], 2016.

LUCIO, J.; VALDÉS, R.; RODRÍGUEZ, L. Improvements to surrogate data methods for nonstationary time series. **Physical Review E**, APS, v. 85, n. 5, p. 056202, 2012.

LYNCH, C. Big data: How do your data grow? **Nature**, Nature Publishing Group, v. 455, n. 7209, p. 28–29, 2008.

MAGRI, C. et al. A toolbox for the fast information analysis of multiple-site lfp, eeg and spike train recordings. **BMC neuroscience**, BioMed Central, v. 10, n. 1, p. 1, 2009.

MALDONADO, Y.; CASTILLO, O.; MELIN, P. Particle swarm optimization of interval type-2 fuzzy systems for fpga applications. **Applied Soft Computing**, Elsevier, v. 13, n. 1, p. 496–508, 2013.

MAO, X.; SHANG, P. Transfer entropy between multivariate time series. **Communications in Nonlinear Science and Numerical Simulation**, Elsevier, v. 47, p. 338–347, 2017.

MAROWKA, A. Python accelerators for high-performance computing. **The Journal of Supercomputing**, Springer, v. 74, n. 4, p. 1449–1460, 2018.

MARX, V. Biology: The big challenges of big data. **Nature**, Nature Research, v. 498, n. 7453, p. 255–260, 2013.

MARZBANRAD, F. et al. Quantifying the interactions between maternal and fetal heart rates by transfer entropy. **PloS one**, Public Library of Science, v. 10, n. 12, p. e0145672, 2015.

MORETTI, L.; SARTORI, L. A simple and resource-efficient setup for the computer-aided drug design laboratory. **Molecular Informatics**, p. n/a–n/a, 2016. ISSN 1868-1751.

NADERI, E. et al. A novel fuzzy adaptive configuration of particle swarm optimization to solve large-scale optimal reactive power dispatch. **Applied Soft Computing**, Elsevier, v. 53, p. 441–456, 2017.

NASROLLAHZADEH, A.; KARIMIAN, G.; MEHRAFSA, A. Implementation of neuro-fuzzy system with modified high performance genetic algorithm on embedded systems. **Applied Soft Computing**, Elsevier, 2017.

Nature Editorial. Reality check on reproducibility. **Nature**, v. 533, p. 437, 2016.

OVADIA, S. Markdown for librarians and academics. **Behavioral & Social Sciences Librarian**, Taylor & Francis, v. 33, n. 2, p. 120–124, 2014.

PÉREZ, F.; GRANGER, B. E. IPython: a system for interactive scientific computing. **Computing in Science and Engineering**, IEEE Computer Society, v. 9, n. 3, p. 21–29, maio 2007. ISSN 1521-9615. Disponível em: <http://ipython.org>.

PERKEL, J. M. Why jupyter is data scientists' computational notebook of choice. **Nature**, v. 563, p. 145–146, 2018.

PIERCE, J. R. **An introduction to information theory: symbols, signals and noise**. [S.l.]: Courier Corporation, 2012.

RAM, K. Git can facilitate greater reproducibility and increased transparency in science. **Source Code for Biology and Medicine**, v. 8, n. 1, p. 7, Feb 2013. ISSN 1751-0473. Disponível em: <https://doi.org/10.1186/1751-0473-8-7>.

REYES-ORTIZ, J. L.; ONETO, L.; ANGUIA, D. Big data analytics in the cloud: Spark on hadoop vs mpi/openmp on beowulf. **Procedia Computer Science**, v. 53, p. 121 – 130, 2015. ISSN 1877-0509.

ROSSUM, G. van; DRAKE, F. L. **The Python Language Reference Manual**. [S.l.]: Network Theory Ltd., 2011. ISBN 1906966141, 9781906966140.

SÁNCHEZ-ORO, J. et al. Improving the performance of embedded systems with variable neighborhood search. **Applied Soft Computing**, Elsevier, v. 53, p. 217–226, 2017.

SANTOS, F. P.; MACIEL, C. D.; NEWLAND, P. L. Pre-processing and transfer entropy measures in motor neurons controlling limb movements. **Journal of computational neuroscience**, Springer, v. 43, n. 2, p. 159–171, 2017.

SCHREIBER, T. Measuring Information Transfer. **Physical Review Letters**, v. 85, n. 2, p. 19, 2000. ISSN 0031-9007.

SCHREIBER, T.; SCHMITZ, A. Improved surrogate data for nonlinearity tests. **Phys. Rev. Lett.**, American Physical Society, v. 77, p. 635–638, Jul 1996.

_____. Surrogate time series. **Physica D: Nonlinear Phenomena**, v. 142, n. 3, p. 346 – 382, 2000. ISSN 0167-2789.

SCHUMAN, C. D. et al. Parallel evolutionary optimization for neuromorphic network training. In: IEEE. **Machine Learning in HPC Environments (MLHPC), Workshop on**. [S.l.], 2016. p. 36–46.

SHANG, F.; CHEN, X.; YAN, C. A strategy for scheduling reduce task based on intermediate data locality of the mapreduce. **Cluster Computing**, Springer, p. 1–11, 2017.

SHANNON, C. A mathematical theory of communication. **The Bell System Technical Journal**, Alcatel-Lucent, v. 27, n. 3, p. 379–423, 1948.

SHAO, S. et al. Accelerating transfer entropy computation. **Proceedings of the 2014 International Conference on Field-Programmable Technology, FPT 2014**, p. 60–67, 2015.

SHARAEV, M.; USHAKOV, V.; VELICHKOVSKY, B. Causal interactions within the default mode network as revealed by low-frequency brain fluctuations and information transfer entropy. In: **Biologically Inspired Cognitive Architectures (BICA) for Young Scientists**. [S.l.]: Springer, 2016. p. 213–218.

SONG, H. et al. Next-generation big data analytics: State of the art, challenges, and future research topics. **IEEE Transactions on Industrial Informatics**, IEEE, 2017.

STERLING, T. et al. Beowulf: A parallel workstation for scientific computation. In: **In Proceedings of the 24th International Conference on Parallel Processing**. [S.l.]: CRC Press, 1995. p. 11–14.

STROHSAL, T. et al. **How Do Financial Cycles Interact? Evidence from the US and the UK**. [S.l.], 2015.

SUN, M. et al. Scheduling algorithm based on prefetching in mapreduce clusters. **Applied Soft Computing**, Elsevier, v. 38, p. 1109–1118, 2016.

TING, T. O. et al. Multicores and gpu utilization in parallel swarm algorithm for parameter estimation of photovoltaic cell model. **Applied Soft Computing**, Elsevier, v. 40, p. 58–63, 2016.

ULLMAN, J. D. Designing good mapreduce algorithms. **XRDS: Crossroads, The ACM Magazine for Students**, ACM, v. 19, n. 1, p. 30–34, 2012.

VENEMA, V.; AMENT, F.; SIMMER, C. A stochastic iterative amplitude adjusted fourier transform algorithm with improved accuracy. **Nonlinear Processes in Geophysics**, Copernicus GmbH, v. 13, n. 3, p. 321–328, 2006.

VICENTE, R. et al. Transfer entropy—a model-free measure of effective connectivity for the neurosciences. **Journal of computational neuroscience**, Springer, v. 30, n. 1, p. 45–67, 2011.

WALT, S. v. d.; COLBERT, S. C.; VAROQUAUX, G. The numpy array: A structure for efficient numerical computation. **Computing in Science and Engg.**, IEEE Educational Activities Department, Piscataway, NJ, USA, v. 13, n. 2, p. 22–30, mar. 2011. ISSN 1521-9615. Disponível em: <<http://dx.doi.org/10.1109/MCSE.2011.37>>.

WEISS, S. M.; INDURKHYA, N. **Predictive data mining: a practical guide**. [S.l.]: Morgan Kaufmann, 1998.

WOLLSTADT, P. et al. Efficient transfer entropy analysis of non-stationary neural time series. **PLoS ONE**, Public Library of Science, v. 9, n. 7, p. 1–21, 07 2014.

YAMAKOV, V. I. **Parallel Grand Canonical Monte Carlo (ParaGrandMC) Simulation Code**. [S.l.], 2016.

YANG, C. et al. Detecting information flow direction in multivariate linear and nonlinear models. **Signal Processing**, Elsevier, v. 93, n. 1, p. 304–312, 2013.

YAO, C.-Z. et al. A study of the transfer entropy networks on industrial electricity consumption. **Entropy**, v. 19, n. 4, 2017.

YAO, Y.; CHANG, J.; XIA, K. A case of parallel eeg data processing upon a beowulf cluster. In: IEEE. **Parallel and Distributed Systems (ICPADS)**, 2009 15th International Conference on. [S.l.], 2009. p. 799–803.

YOOK, S.-H. et al. Finding modules and hierarchy in weighted financial network using transfer entropy. **Physica A: Statistical Mechanics and its Applications**, Elsevier, v. 447, p. 493–501, 2016.

ZHAI, L.-S. et al. The measurement of gas–liquid two-phase flows in a small diameter pipe using a dual-sensor multi-electrode conductance probe. **Measurement Science and Technology**, v. 27, n. 4, p. 045101, 2016. Disponível em: <<http://stacks.iop.org/0957-0233/27/i=4/a=045101>>.

ZHU, J.; GE, Z.; SONG, Z. Distributed parallel pca for modeling and monitoring of large-scale plant-wide processes with big data. **IEEE Transactions on Industrial Informatics**, IEEE, 2017.

Appendix

APPENDIX A – SUBMITTED PAPERS

A.1 COMPLEXIS 2017

Submitted paper "Parallelism Strategies for Neurophysiological Delayed Transfer Entropy Data Processing: Towards Causal Inference in Big Data" was **accepted** by International Conference on Complexity, Future Information Systems and Risk (COMPLEXIS) 2017 as shown [Figure 18](#). Paper first page was appended in [Page 76](#).

Institute for Systems and Technologies of Information, Control and Communication

COMPLEXIS 2017

2nd International Conference on Complexity, Future Information Systems and Risk
<http://www.complexis.org/>

To whom it may concern,

We are happy to inform that the paper submitted by Jonas Rossi Dourado, José Roberto B. de A. Monteiro, Daniel Rodrigues de Lima, Michel Bessani and Carlos Dias Maciel to COMPLEXIS 2017 with number 7, entitled "Parallelism Strategies for Neurophysiological Delayed Transfer Entropy Data Processing", has been accepted as a Full Paper, to be presented April (24 - 26) at Porto, Portugal.

All papers accepted to COMPLEXIS 2017 were peer reviewed by at least two experts from the international program committee, in a double-blind review process. The paper will be published in the conference proceedings with up to 12 pages, and after being presented at COMPLEXIS 2017 it will be included in the SCITEPRESS Digital Library under a specific DOI to be specified after the proceedings are published, and submitted for indexation to Thomson Reuters Conference Proceedings Index, Engineering Index, DBLP, INSPEC and Scopus.

Best Regards,

Victor Chang
 (COMPLEXIS Conference Chair)

HQ: Rua dos Lusíadas, 118 3º Frente, 1300-376 Lisboa, Portugal
 Office: Av. D. Manuel I, 27A - 2º Esq., 2910-595 Setúbal, Portugal
 E-mail: secretariat@insticc.org web: <http://www.insticc.org>

Figure 18: Acceptance Letter from COMPLEXIS 2017

Parallelism Strategies for Neurophysiological Delayed Transfer Entropy Data Processing: Towards Causal Inference in Big Data

Jonas Rossi Dourado¹, José Roberto B. de A. Monteiro¹, Daniel Rodrigues de Lima¹,
Michel Bessani¹ and Carlos Dias Maciel¹

¹*Department of Electrical and Computational Engineering, University of São Paulo, São Carlos, São Paulo, Brazil*
{jonas.dourado, carlos.maciel}@usp.br

Keywords: Delayed Transfer Entropy. Neurophysiological Data. Causality. Parallelism Strategies. Big Data Analysis. Task Parallelism. Data Parallelism, Data Analysis Performance.

Abstract: Nowadays, the amount of data being generated and collected has been rising with the popularization of technologies such as Internet of Things, social media, and smartphone. The increasing amount of data led the creation of the term big data. One class of Big Data hidden information is causality. Among the tools to infer causal relationships there is Delayed Transfer Entropy (DTE); however, it has a high demanding processing power. Many approaches were proposed to overcome DTE performance issues such as GPU and FPGA implementations. Our approach is to compare different parallel strategies to calculate DTE from neurophysiological time series using a heterogeneous Beowulf cluster aiming to increase DTE performance.

1 INTRODUCTION

Nowadays, the amount of data being generated and collected has been rising with the popularization of technologies such as Internet of Things, social media, and smartphone (Hashem et al., 2015). The increasing amount of data led the creation of the term big data, with one definition given by Hashem et al. (2015), as a set of technologies and techniques to discover hidden information from diverse, complex and massive scale datasets. One class of hidden information is causality, which Bareinboim & Pearl (2016) discuss and propose a framework to deal with common found big data biases such as confounding and sampling selection.

Among the tools to infer causal relationships there are Mutual Information used by Endo et al. (2015) to infer neuron connectivity and Granger causality used by Strohsal et al. (2015) to model causality between US and UK economies. Additionally, exist Transfer Entropy (TE), which allows identification of cause-effect relationship by not accounting for simple and uniquely shared information (Schreiber, 2000). TE has been applied to many problems from diverse research fields e.g. finance (Yook et al., 2016); biosignals (Marzbanrad et al., 2015), complex networks (Haruna & Fujiki, 2016) and climatology (Hi-

rata et al., 2016).

A derivation of TE metric called Delayed TE (DTE) is useful for neurophysiological causality as used by Ito et al. (2011) to identify active connections between neurons and by Wollstadt et al. (2014) to calculate information transfer and delays from magnetoencephalography signals. Despite TE and DTE wide applicability, they have a high demanding processing power (Shao et al., 2015), which is aggravated with large datasets as those found in big data. Many approaches were proposed to overcome performance issues such as an implementation using a GPU made by Wollstadt et al. (2014) and an implementation using an FPGA made by Shao et al. (2015). Another approach to speedup data analysis is using a computer cluster.

Parallel programs should be optimized to extract maximum performance from hardware on architecture case by case, which is far from trivial according to Booth et al. (2016). There exist different and combined manners to explore parallelism such as data parallelism and task parallelism (Gordon et al., 2006). Choudhury et al. (2015) stated that choosing the configuration of parallel programs is a "mysterious art" in a study which they created a model aiming maximum speedup by balancing different parallelism strategies for both cluster and cloud computer environments.

A.2 Applied Soft Computing

After COMPLEXIS 2017, we received an invite to Applied Soft Computing Journal fast track process. Submitted paper first page is shown in Page [78](#).

The paper that you have presented at COMPLEXIS 2017 with the title "Parallelism Strategies for Neurophysiological Delayed Transfer Entropy Data Processing: Towards Causal Inference in Big Data" has been invited to submit an extended version (at least 30% of new content) to be appreciated for publication in a Fast Track submission process of the Elsevier Applied Soft Computing (ASOC) Journal.

2

4

6

7

Keywords: Delayed Transfer Entropy, Parallelism Strategies, Big Data Analysis, Heterogeneous Computer Cluster, Causality, Surrogate

10

18

A.3 Congresso Brasileiro de Automática 2018

Accepted paper to COMPLEXIS 2017 was resubmitted to Congresso Brasileiro de Automática (CBA) 2018 and **published**.

PARALLELISM STRATEGIES FOR NEUROPHYSIOLOGICAL DELAYED TRANSFER ENTROPY DATA PROCESSING: TOWARDS CAUSAL INFERENCE IN BIG DATA

JONAS ROSSI DOURADO*, MICHEL BESSANI*, DANIEL RODRIGUES DE LIMA*, JOSÉ ROBERTO B. DE A. MONTEIRO*, RAFAEL RODRIGUES MENDES RIBEIRO*, CARLOS DIAS MACIEL*

**Department of Electrical and Computational Engineering,
University of São Paulo,
São Carlos, São Paulo, Brazil*

Emails: jonas.dourado@usp.br, michel.bessani@gmail.com, drodriguesdelima@gmail.com, jrm@sc.usp.br, rafael.mendes.ribeiro@usp.br, maciel@sc.usp.br

Abstract— Nowadays, the amount of data being generated and collected has been rising with the popularization of technologies such as Internet of Things, social media, and smartphone. The increasing amount of data led the creation of the term big data. One class of Big Data hidden information is causality. Among the tools to infer causal relationships there is Delayed Transfer Entropy (DTE); however, it has a high demanding processing power. Many approaches were proposed to overcome DTE performance issues such as GPU and FPGA implementations. Our approach is to compare different parallel strategies to calculate DTE from neurophysiological time series using a heterogeneous Beowulf cluster aiming to increase DTE performance.

Keywords— Delayed Transfer Entropy. Neurophysiological Data. Causality. Parallelism Strategies. Big Data Analysis. Task Parallelism. Data Parallelism, Data Analysis Performance.

1 Introduction

Nowadays, the amount of data being generated and collected has been rising with the popularization of technologies such as Internet of Things, social media, and smartphone (Hashem et al., 2015). The increasing amount of data led the creation of the term big data, with one definition given by (Hashem et al., 2015), as a set of technologies and techniques to discover hidden information from diverse, complex and massive scale datasets. One class of hidden information is causality, which (Bareinboim and Pearl, 2016) discuss and propose a framework to deal with common found big data biases such as confounding and sampling selection.

Among the tools to infer causal relationships there are Mutual Information used by (Endo et al., 2015) to infer neuron connectivity and Granger causality used by (Strohsal et al., 2015) to model causality between US and UK economies. Additionally, exist Transfer Entropy (TE), which allows identification of cause-effect relationship by not accounting for simple and uniquely shared information (Schreiber, 2000). TE has been applied to many problems from diverse research fields e.g. finance (Yook et al., 2016); biosignals (Marzbanrad et al., 2015), complex networks (Haruna and Fujiki, 2016) and climatology (Hirata et al., 2016).

A derivation of TE metric called Delayed TE (DTE) is useful for neurophysiological causality as used by (Ito et al., 2011) to identify active connections between neurons and by (Wollstadt et al., 2014) to calculate information transfer and delays from magnetoencephalography signals. Despite TE and DTE wide applicability, they have a high demanding processing power (Shao et al., 2015), which is aggravated with large datasets as those found in big data. Many approaches were proposed to overcome performance issues such as an implementation using a GPU made

by (Wollstadt et al., 2014) and an implementation using an FPGA made by (Shao et al., 2015). Another approach to speedup data analysis is using a computer cluster.

Parallel programs should be optimized to extract maximum performance from hardware on architecture case by case, which is far from trivial according to (Booth et al., 2016). There exist different and combined manners to explore parallelism such as data parallelism and task parallelism (Gordon et al., 2006). (Choudhury et al., 2015) stated that choosing the configuration of parallel programs is a "mysterious art" in a study which they created a model aiming maximum speedup by balancing different parallelism strategies for both cluster and cloud computer environments.

In this study, we compare parallel strategies to calculate DTE from neurophysiological time series using a heterogeneous Beowulf cluster aiming to increase DTE performance. We also analyze computing node performance within task parallelism to gain some insights to enrich parallel strategies discussion.

This paper is organized as follows: Until introduction end, it will be presented concepts that might help readers keep up with whole paper. In materials and methods, will be described all steps needed to reproduce the results. Results and discussion are self-explanatory. In conclusion, additionally, is suggested future works. Remaining information useful to reproducibility is located in an appendix to avoid nonessential noise through the text.

1.1 Beowulf Cluster

A Beowulf cluster is made by connecting consumer grade computers on a local network using Ethernet or other suitable connection technology (Yao et al., 2009). The term Beowulf cluster was coined by (Sterling et al., 1995), which created the topology on

A.4 Entropy journal

We improved our extended version submitted to Applied Soft Computing and resubmitted to Entropy journal. The result was "Declined for Publication - Encourage Resubmission after Revisions". We intend to submit the extended version to arXiv

Article

Parallelism Strategies for Big Data Delayed Transfer Entropy

Jonas Rossi Dourado ^{1,†}, Michel Bessani ^{2,‡}, José Roberto Monteiro ^{3,†}, Tadeu Junior Gross ^{4,†}, William Darwin Junior ^{5,†} and Carlos Dias Maciel ^{6,*}

[†] Department of Electrical and Computational Engineering, University of São Paulo, São Carlos, São Paulo, Brazil;

[‡] Department of Electrical Engineering, Federal University of Minas Gerais

¹ jonas.dourado@usp.br

² mbessani@ufmg.br

³ jrm@sc.usp.br

⁴ tadeugross@politec.mt.gov.br

⁵ wdarwinjr@gmail.com

* Correspondence: maciel@sc.usp.br;

Academic Editor: name

Version September 21, 2018 submitted to Entropy

Abstract: Generated and collected data has been rising with the popularization of technologies such as Internet of Things, social media, and smartphone, leading big data term creation. One class of big data hidden information is causality. Among the tools to infer causal relationships there is Delay Transfer Entropy (DTE); however, it has a high demanding processing power. Many approaches were proposed to overcome DTE performance issues such as GPU and FPGA implementations. Our study compared different parallel strategies to calculate DTE from big data series using a heterogeneous Beowulf cluster. Task Parallelism was significantly faster in comparison to Data Parallelism. With big data trend in sight, this results may enable bigger datasets analysis or better statistical evidence.

Keywords: Delayed Transfer Entropy; Parallelism Strategies; Big Data Analysis; Heterogeneous Computer Cluster; Complex Systems; Causality; Surrogate

1. Introduction

Recently, the amount of data being generated and collected has been rising with the popularization of technologies such as Internet of Things, social media, and smartphone [1]. The increasing amount of data led the creation of the term big data, with one definition given by Hashem *et al.* [1], as a set of technologies and techniques to discover hidden information from diverse, complex and massive scale datasets. One class of hidden information is causality, which Bareinboim and Pearl [2] discuss and propose a framework to deal with commonly found big data biases such as confounding and sampling selection.

The interaction among random variables and subsystems in complex dynamical multivariate models as seen in big data is a developing research area that had been widely applied in a variety of fields, such as climatic processes [3,4], brain analysis [5], pathology propagation [6], circadian networks [7], among others. The knowledge about causality in such complex systems can be useful for the evaluation of their dynamics and also to their modeling since it can lead to topological simplifications. In many cases, the assessment of interaction or coupling is subject to bias [8].

Among the tools to infer causal relationships, there are Mutual Information used by Endo *et al.* [9] to infer neuron connectivity or Granger causality employed by Li *et al.* [10] to detect industrial process fault root cause using data analysis. Additionally, exists Transfer Entropy (TE), which allows

APPENDIX B – PLOT PART II - TRANSFER ENTROPY SERIAL OPTIMIZATION

Listing B.1: [Figure 12](#) source code

```

1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 Created on Thu Aug 31 03:58:05 2017
5
6 @author: jonaías
7 """
8
9 import matplotlib.pyplot as plt
10 import numpy as np
11
12
13 data1 = [53.54414200782776,53.69987750053406,54.15251898765564,
14          54.23860740661621,54.81629776954651,53.69047808647156,56.46116828918457,
15          55.98073840141296,56.42609739303589,55.51141095161438]
16
17 data2 =[19.46474051475525,19.41711711883545,19.71887683868408,
18          19.82204127311706,19.87676024436950,19.52250742912292,20.63817477226257,
19          20.25014042854309,20.50941205024719,20.25551795959472]
20
21
22 plt.figure()
23
24 plt.ylabel('DTEinnerloopexecutiontime(s)')
25
26 plt.boxplot([data1,data2], labels=["CODEA", "CODEB(OptimizedDTE)"])
27 plt.grid()
28 plt.savefig('serial_optimization.pdf')
29
30
31 mean_a = np.mean(data1)
32 mean_b = np.mean(data2)
33
34 print("Speedup:  $\frac{{}}{}} = {}$ ".format(mean_a,mean_b,mean_a/mean_b))

```


APPENDIX C – DOCUMENTATION

This guide shows how to install a jupyter enabled Beowulf cluster.

C.1 Cluster installation

Install Fedora 26 on all machines

Cluster nodes are computation nodes and host is where jupyter notebook is executed and has connection to external network.

Cluster nodes

Download and install Fedora Server 26 (or newer) using image ISO

<https://getfedora.org/en/server/download/>

Follow Complete installation guide, topic "Installing in the Graphical User Interface"

https://docs.fedoraproject.org/f26/install-guide/install/Installing_Using_Anaconda.html

1. Set installation device priority to any available SSD, with automatically partitioning
2. Set node hostname under "NETWORK & HOSTNAME". We named as lpsXX, with XX being an incremental number
3. Set network to ip to automatic (DHCP) under "NETWORK & HOSTNAME"
4. Set root password and create an user with admin privileges

Host

Download and install Fedora Workstation 26 (or newer) using image ISO

<https://getfedora.org/en/workstation/download/>

Follow Complete installation guide, topic "Installing in the Graphical User Interface"

https://docs.fedoraproject.org/f26/install-guide/install/Installing_Using_Anaconda.html

1. Set installation device priority to any available SSD, with automatically partitioning
2. Set node hostname under "NETWORK & HOSTNAME"
3. Set network to ip to automatic (DHCP) under "NETWORK & HOSTNAME"
4. Set root password and create an user with admin privileges
5. Add to /etc/hosts the following lines. Change machine naming (lpsXX) according machine hostnames

192.168.1.100	lps00
192.168.1.101	lps01
192.168.1.102	lps02
192.168.1.103	lps03
192.168.1.104	lps04
192.168.1.105	lps05
192.168.1.106	lps06
192.168.1.107	lps07
192.168.1.108	lps08
192.168.1.109	lps09
192.168.1.110	lps10
192.168.1.111	lps11
192.168.1.112	lps12

6. Set host ips (enable external access)

Network "enp3s0" has a external static ip.

Network "eno1" has a dynamic ip set by cluster router DHCP server


```
[lps@lps00 ~]$ cat /etc/sysconfig/network-scripts/ifcfg-enp3s0
TYPE="Ethernet"
BOOTPROTO=none
DEFROUTE="yes"
IPV4_FAILURE_FATAL="no"
IPV6INIT=no
IPV6_AUTOCONF="yes"
IPV6_DEFROUTE="yes"
IPV6_FAILURE_FATAL="no"
NAME="enp3s0"
UUID="271c18ee-c5be-4223-b72e-68c20178d38e"
ONBOOT="yes"
DNS1=143.107.225.6
DNS2=143.107.182.2
IPV6_PEERDNS=yes
IPV6_PEERROUTES=yes
HWADDR=00:E0:53:14:50:B2
IPADDR=143.107.235.60
PREFIX=24
GATEWAY=143.107.235.1
```

```
[lps@lps00 ~]$ cat /etc/sysconfig/network-scripts/ifcfg-eno1
TYPE="Ethernet"
BOOTPROTO=none
DEFROUTE="yes"
IPV4_FAILURE_FATAL="no"
IPV6INIT="yes"
IPV6_AUTOCONF="yes"
IPV6_DEFROUTE="yes"
IPV6_FAILURE_FATAL="no"
NAME="eno1"
UUID="e849f50e-6006-4d3c-b39d-ca2299c9be24"
ONBOOT=yes
BROADCAST=192.168.1.255
NETWORK=192.168.1.0

HWADDR=E0:69:95:68:9E:9C
IPADDR=192.168.1.100
PREFIX=24
GATEWAY=192.168.1.1
IPV6_PEERDNS=yes
IPV6_PEERROUTES=yes

ETHTOOL_OPTS="wol g autoneg off speed 100 duplex full"
```

```
[lps@lps00 ~]$ ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: enp3s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
group default qlen 1000
    link/ether 00:e0:53:14:50:b2 brd ff:ff:ff:ff:ff:ff
    inet 143.107.235.60/24 brd 143.107.235.255 scope global enp3s0
        valid_lft forever preferred_lft forever
    inet6 fe80::2e0:53ff:fe14:50b2/64 scope link
        valid_lft forever preferred_lft forever
3: eno1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
group default qlen 1000
    link/ether e0:69:95:68:9e:9c brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.100/24 brd 192.168.1.255 scope global eno1
        valid_lft forever preferred_lft forever
    inet6 fe80::e269:95ff:fe68:9e9c/64 scope link
        valid_lft forever preferred_lft forever
[lps@lps00 ~]$ route
Tabela de Roteamento IP do Kernel
Destino          Roteador      MáscaraGen.   Opções Métrica Ref  Uso Iface
default          gateway       0.0.0.0       UG      100    0      0 enp3s0
default          gateway       0.0.0.0       UG      101    0      0 eno1
143.107.235.0    0.0.0.0       255.255.255.0 U        100    0      0 enp3s0
192.168.1.0      0.0.0.0       255.255.255.0 U        100    0      0 eno1
192.168.122.0    0.0.0.0       255.255.255.0 U         0      0      0 virbr0
```

7. Enable ssh server

```
sudo systemctl enable sshd
```

Configure router DHCP server on wireless router

DHCP server should set fixed IP according to MAC Address. To do so, we configure reserved ip on router

1. Go to <http://192.168.1.1/>. Login: admin Pass:admin
2. DHCP -> Address reservation
3. Set MAC Address and IP association to every computer

Install software on cluster nodes and cluster host

```
# Check for new versions
sudo dnf check-update
# Update all packages
sudo dnf update --assumeyes
# Install pip and other useful packages
sudo dnf install tmux htop python3-pip vim gcc redhat-rpm-config python3-devel
fftw-devel --assumeyes
# Install Python packages
sudo pip3 install jupyter pyfftw ipyparallel scipy seaborn matplotlib
```

Configure login permissions

1. Generate ssh key on cluster host

```
ssh-keygen
```

2. Copy cluster host key to every cluster node. Use lps01, lps02, ... to lps12

```
ssh-copy-id lps@<node>
```

3. Disable remote login with password on cluster host. Allow only public-private key login.

Create ipyparallel cluster config

1. Enable ipyparallel cluster extension to jupyter-notebook

```
ipcluster nbextension enable
```

2. Create profile with on cluster host

```
ipython profile create --parallel --profile=myprofile
```

3. Edit file '/home/lps/.ipython/profile_lpscluster/ipcluster_config.py' with:

```
c.IPClusterEngines.engine_launcher_class = 'SSH'
c.IPClusterStart.controller_ip = '192.168.1.100'
c.IPClusterStart.controller_launcher_class = 'SSH'
c.SSHControllerLauncher.hostname = '192.168.1.100'
c.SSHControllerLauncher.user = 'lps'
c.SSHEngineSetLauncher.engines = {'lps01' : 8, 'lps02' : 8, 'lps03' : 8,
'lps04' : 8, 'lps05' : 8, 'lps06' : 8, 'lps07' : 8, 'lps08' : 8, 'lps09' : 8,
'lps10' : 8, 'lps11' : 8, 'lps12' : 8}
```

4. Add password to jupyter notebook

```
jupyter notebook password
```

Share folder /experiments across cluster

On cluster host

1. Install and enable nfs server

```
dnf -y install nfs-utils
systemctl start rpcbind nfs-server
systemctl enable rpcbind nfs-server
firewall-cmd --add-service=nfs --permanent
```

2. Create folder /experiment to allow mounting

```
sudo mkdir /experiments
```

3. Append to /etc/fstab secondary HD as /experiment

```
UUID=572bad1d-54fd-45cb-bd18-ef273542218e /experiments ext4
defaults 1 1
```

4. Share experiments on nfs (append to /etc/exports)

```
/experiments *(rw,sync,no_root_squash)
```

On cluster nodes

1. Create folder /experiment to allow mounting

```
sudo mkdir /experiments
```

2. Append to /etc/fstab nfs share

```
192.168.1.100:/experiments /experiments nfs
rsize=8192,wsz=8192,timeo=14,_netdev 0 0
```

Start jupyter-notebook on cluster host start

Set configurations on

/home/lps/.jupyter/jupyter_notebook_config.py

```
c.NotebookApp.notebook_dir = '/experiments/'
c.NotebookApp.open_browser = False
```

Create file /usr/lib/systemd/system/jupyter.service

```
[Unit]
Description=Jupyter Notebook

[Service]
Type=simple
PIDFile=/run/jupyter.pid
ExecStart=/usr/bin/jupyter-notebook --
config=/home/lps/.jupyter/jupyter_notebook_config.py
User=lps
WorkingDirectory=/experiments/
Restart=always
RestartSec=10

[Install]
WantedBy=multi-user.target
```

Reload and enable

```
sudo systemctl enable jupyter.service
sudo systemctl daemon-reload
sudo systemctl restart jupyter.service
```

C.2 *ipyparallel* usage example

To log on Jupyter, go to url:

External access: `http://143.107.235.60:8888/login`

Internal access: `http://192.168.1.100:8888/login`

Password: `lablpstop`

How to use

Each cluster user should exclusively use its own folder. If not created on jupyter root, the user must create.

tip: All folders **in** jupyter root are mapped across whole cluster, within same folder (`/experiments/`).

If any **user** wants **to** upload data, the **user** may upload using ssh **to** master node `/experiments/<username>/`

The easiest way to use whole cluster is loop parallelization with *ipyparallel* `map_sync`. The function to execute processing itself may open/write different files, according to each parameter.

```

import ipyparallel as ipp
rc = ipp.Client(profile='lpscluster')
dview = rc[:] # use all engines

# Display all nodes that will be used in calculations
# print(rc.ids)

intervals = [(20,100), (200,500), (1000,2000)]

data = np.ones((10000))

# Note that volatility is executed only on cluster nodes,
# not on host machine, so numpy needs only to be imported on nodes
def volatility(samples):
    return np.sum(samples)

# This function gets each individual interval as partitionates data, so we can
# execute volatility to each partitioned data
def execute_volatility(interval):
    return volatility(data[interval[0]:interval[1]])

# Imports should be manually done in all nodes -
dview.execute('import numpy as np')

# Send variables to all nodes
dview['data'] = data
dview['volatility'] = volatility

# Execute map, it distributes each interval across nodes automagically =)
n_parallel = dview.map_sync(execute_volatility, intervals)

# Display result.. for example, we can save results on file instead of only
# displaying them
print(n_parallel)

```

To start cluster DTE processing using LPSLIB

All commands should be executed on master node.

Go to lpslib folder. eg:

```
cd /experiments/jonas/lpslib
```

Start IPython cluster:

```
ipcluster start --profile=lpscluster
```

Start log gather (Completely optional, will gather logs from all cluster nodes)

```
python3 log_gather.py
```

Start executable

```
python3 ExecDTE.py -i dados.execucao.maciел --nsur 35 -ipyprofile lpscluster --  
log_ip=192.168.1.100
```


C.3 Ipslib documentation

Ipslib

Signal Processing Libraries developed by LPS. All these functions support performing Delayed Transfer Entropy (DTE) analysis. The library has methods for calculating Delayed and non-Delayed Transfer Entropy (*DTE_{xy}*, *TE_{xy}*), Delayed and non-Delayed Mutual Information (*DMI_{xy}*, *MI_{xy}*), signal surrogate (*AAFT*) and stationary test (*wwRunTest*).

1 - Library architecture

The library is composed by Information Theory measurements / stationary test (*SinaisMedidas.py*) and surrogate generation (*SinaisSurrogate.py*) modules.

Additionally, the library comes with an application example that follows the same methodology as "[Santos, Fernando P., Carlos D. Maciel, and Philip L. Newland. "Pre-processing and transfer entropy measures in motor neurons controlling limb movements." *Journal of computational neuroscience* 43.2 \(2017\): 159-171 \(https://doi.org/10.1007/s10827-017-0656-6\)](https://doi.org/10.1007/s10827-017-0656-6)". The application is divided in DTE calculation (*DTEexec.py*) and analysis (*DTEanalysis.py*). Both *DTEexec.py* and *DTEanalysis.py* uses signal reading (**SignalReader.py*) abstraction modules.

2 - Application example

The application example uses folder `./output` to save results. When calling *DTEexec.py* or *DTEanalysis.py* target folder should be specified with parameter `-t <target folder>` so result will be saved under `./output/<target folder>`

Calculating DTE

`ipyparallel` library is employed to decrease DTE run time, by parallellizing *DTEexec.py* code across processor cores/cluster nodes. First we need to start `ipyparallecluster` manager:

```
ipcluster start -n <num cores> # Using only local processor cores e.g. user laptop
OR
ipcluster start --profile=lpscluster # On cluster host machine, to use 'lpscluster' ipyparallel cluster
```

Once cluster is running, the user can start DTE calculation by:

```
python3 DTEexec.py -i run.neuron --nsur 35 # Using only local processor cores e.g. user laptop
OR
python3 DTEexec.py -i run.neuron --nsur 35 -ipyprofile lpscluster -t <target folder> --log_ip=192.168.1.100 # On cluster host machine, to use 'lpscluster' ipyparallel cluster
```

The default signal reader is *NeuronSignalReader*, which read signals from folder `./data/neuron`. The *DTEexec.py* outputs are:

1. Folder `./output/<target folder>/embed`: Embedding values
2. Folder `./output/<target folder>/dte`: Pickled DTE results of signal (`<signal name>_<channel x>_<channel y>_REF.dte`) and its surrogates (`<signal name>_<channel x>_<channel y>_SUR.dte`, with all surrogates in a single file)
3. folder `./output/<target folder>/log`: Log files

The format of input file is a single signal on each line. `DTEexec.py` permutes every signal channel and executes DTE on each pair.

```
<signal name> <start sample> <stop sample>
```

Additional `DTEexec.py` parameters are shown by issuing `--help`. The most important are `--lag`, `--nbins` and `--nsur`

```
-h, --help            show this help message and exit
-i INPUT_FILE, --input_file INPUT_FILE
                        file containing signals to be executed.
                        default='run.neuron'
--lag LAG              Maximum lag size in milliseconds. Default=105
--nbins NBINS          Number of bins used to generate pdf. Default=5
--nsur NSUR            Number of generated surrogate. Default=1
--log_ip LOG_IP        Log gather ip. Default='127.0.0.1'
--ipyprofile IPYPROFILE
                        ipyparallel profile name. Default='default'
-r READER, --reader READER
                        Signal reader name (e.g. NeuronSignalReader,
                        SimulatedSignalReader, ...)
                        Default='NeuronSignalReader'
-t TARGET_DIR, --target_dir TARGET_DIR
                        Target folder under ./output/. Default='target'
```

DTE Analysis

To generate DTE plots, just execute:

```
python3 DTEAnalysis.py -i run.neuron -t <target folder>
```

The default signal reader is `NeuronSignalReader`, which reads signals from folder `.data/neuron`. The `DTEAnalysis.py` outputs are:

1. Folder `./output/<target folder>/dte_plot`: DTE plots

The format of input file is a single signal on each line. `DTEAnalysis.py` creates a `.pdf` file with a plot for each signal permutation.

```
<signal name> <start sample> <stop sample>
```

How to create a signal reader

Signal reader can be created by inheriting `BaseSignalReader` class. The methods that should be implemented are `get_data` and `get_metadata`, which can be easily inspired from `SimulatedSignalReader.py`.

3 - For legacy users

1. dados.execucao renamed to run.<signal reader name>
2. dados.descricao moved to ./data/neuron/neuron.metadata
3. Data files moved from ../Dados to ./data/<signal reader name>/
4. Output are written under folder ./output
5. ExecDTE.py renamed to DTEexec.py
6. TEAnalysis.py renamed to DTEAnalysis.py
7. TEAnalysis now uses same input file as DTEexec.py
8. Legacy scripts were moved to ./legacy folder

4 - Dependencies

1. Install using OS package manager python3, fftw3, pip3
2. sudo pip3 install scipy ipyparallel matplotlib pyfftw peakutils