UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação

Data analysis over large-scale graphs using vertex-centric asynchronous parallel processing

Gabriel Perri Gimenes

Tese de Doutorado do Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional (PPG-CCMC)

ICMC USP SÃO CARLOS

Gabriel Perri Gimenes

Data analysis over large-scale graphs using vertex-centric asynchronous parallel processing

Thesis submitted to the Institute of Mathematics and Computer Sciences – ICMC-USP – in accordance with the requirements of the Computer and Mathematical Sciences Graduate Program, for the degree of Doctor in Science. *FINAL VERSION*

Concentration Area: Computer Science and Computational Mathematics

Advisor: Prof. Dr. José Fernando Rodrigues Jr.

USP – São Carlos
April 2020

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados inseridos pelo(a) autor(a)

P456d

Perri Gimenes, Gabriel
Data analysis over large-scale graphs using vertex-centric asynchronous parallel processing / Gabriel Perri Gimenes; orientador José Fernando Rodrigues Jr. -- São Carlos, 2020.
89 p.

Tese (Doutorado - Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2020.

1. graph processing. 2. fraud detection. 3. belief propagation. 4. vertex-centric algorithms. 5. asynchronous processing. I. Rodrigues Jr, José Fernando , orient. II. Título.

Bibliotecários responsáveis pela estrutura de catalogação da publicação de acordo com a AACR2:
Gláucia Maria Saia Cristianini - CRB - 8/4938
Juliana de Souza Moraes - CRB - 8/6176

Gabriel Perri Gimenes

Análise de dados sobre grafos em larga escala por meio de processamento paralelo assíncrono centrado em vértices

Tese apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Doutor em Ciências – Ciências de Computação e Matemática Computacional. *VERSÃO REVISADA*

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientador: Prof. Dr. José Fernando Rodrigues Jr.

USP – São Carlos
Abril de 2020

ACKNOWLEDGEMENTS

I would like to thank the University of São Paulo (USP), and the *Instituto de Ciências Matemáticas e de Computação* (ICMC) for providing an excellent environment for the development of my thesis. I would also like to thank the Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), process number 2014/25337-0 and 2016/02557-0; CNPq process number 305580/2017-5; as well as the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) for their financial support during the development of this project.

To my advisor, Prof. José Fernando Rodrigues Jr., for his guidance, knowledge, time, and most importantly for his patience during the development of this project and throughout the whole journey. To all my colleagues and fellow students.

I would also like to give special thanks to my fiancée Gabrielle Alvim, for her invaluable partnership. And finally, I would like to thank my family for everything they have done and continue to do to support me.

“And, in the end
The love you take
is equal to the love you make”
Paul McCartney

ABSTRACT

GIMENES, GABRIEL P. **Data analysis over large-scale graphs using vertex-centric asynchronous parallel processing**. 2020. 89 p. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2020.

Since the birth of web 2.0, users no longer just consume but are now active creators of content that is going to be consumed by other users. This new dynamic took data generation to a whole new scale, called planetary-scale or web-scale. Often, this data represents relationships between its elements, such as in social networks, recommendation systems, online boards, email networks, scientific citation networks, and others. Analyzing how information flows and how nodes influence each other in several of such networks is a widely regarded problem; While Belief Propagation, which is the fundamental algorithm for these types of inference, is widely used, it historically lacked convergence guarantees for real-world networks. However, even though recently alternative methods such as LinBP solve the convergence problems of the original algorithm, it's scalability when dealing with large-scale problems remains a challenge. Also, several of the works proposed to solve this issue, do so by relying on specific infrastructures such as supercomputers and computational clusters. Motivated by these challenges we propose a new algorithm, called VCBP, that aims to provide a scalable framework for belief propagation on large-scale problems, such as when graphs do not fit the main memory. We do so by combining state-of-the-art asynchronous vertex-centric parallel processing with state-of-the-art belief propagation algorithm. Our algorithm maintains the same accuracy rate while achieving performance orders of magnitude higher than former LinBP's implementation. Due to the asynchronous nature of our algorithm, VCBP demands fewer iterations before convergence than any previous algorithm. Additionally, we analyze our algorithm in the task of node classification, achieving significant results over real-world datasets. Our findings indicate that there is unexplored potential in today's widely available modern hardware, specifically concerning parallelism, sparking a shift towards a more cost-efficient and ubiquitous data mining scenario.

Keywords: graph processing, belief propagation, parallel processing, vertex-centric processing, big data.

RESUMO

GIMENES, GABRIEL P. **Análise de dados sobre grafos em larga escala por meio de processamento paralelo assíncrono centrado em vértices**. 2020. 89 p. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2020.

Desde o surgimento da web 2.0, os usuários não mais apenas consomem conteúdo, mas também são responsáveis agora por criar conteúdo que será consumido por outros usuários. Essa nova dinâmica levou a produção de dados à uma nova e surpreendente escala, chamada de escala planetária. Muitas vezes tais dados representam relacionamentos entre seus elementos, como é o caso em redes sociais, sistemas de recomendação, fórums online, redes de email, redes de citação científica, entre outras. Analisar o fluxo de informações e como os nós influenciam uns aos outros nesses domínios é um problema recorrente; Apesar do algoritmo *Belief Propagation* ser um dos principais algoritmos utilizados nesse contexto, o algoritmo historicamente apresentou problemas de garantias de convergência quando aplicado à redes reais. Contudo, apesar de recentemente métodos alternativos como LinBP focarem em resolver o problema de convergência do algoritmo original, a escalabilidade do algoritmo em grafos de larga escala continua sendo um desafio. Além disso, muitas das propostas que tentam resolver o problema de escalabilidade necessitam de infraestrutura adicional como supercomputadores e clusters computacionais. Com a motivação desses desafios, essa tése propoe um novo algoritmo, chamado VCBP, que tem como objetivo prover um arcabouço escalável para *Belief Propagation* em problemas de larga escala, como ocorre nos casos em que o grafo não cabe na memória principal. A proposta combina técnicas de processamento paralelo assíncrono centrado em vértices com avanços de estado-da-arte no algoritmo de *Belief Propagation*. O VCBP é capaz de alcançar novos patamares de performance que são ordens de magnitude melhores que a implementação do LinBP. Além disso, devido à natureza assíncrona do algoritmo são necessárias menos iterações até que a convergência seja alcançada quando comparado com outras soluções. Por fim, analisamos também o algoritmo quando aplicado à tarefa de classificação, alcançando resultados significativos em bases de dados reais. Nossas descobertas indicam que existe um grande potencial inexplorado na tecnologia de hardware largamente disponível atualmente, especialmente em relação ao paralelismo, apontando para a oportunidade de uma computação mais acessível e com melhor custo-benefício.

Palavras-chave: processamento de grafos, propagação de crenças, processamento paralelo, processamento centrado em vertíces, larga escala.

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF TABLES

LIST OF SYMBOLS

G — graph

V — vertex-set

E — edge-set

C — class-set

$\mathbf{e_s}$ — normalized vector of explicit (priors) beliefs of node s

$\mathbf{b_s}$ — normalized vector of implicit (posterior) beliefs of node s

c — number of classes (beliefs)

$V_{Explicit}(v)$ — explicit belief values for a vertex v

H — coupling matrix between classes

$H(\text{from}, \text{to})$ — coupling strength between those classes from and to

$m_{st}, \hat{m}_{st}$ — messages sent from vertex s to t

$b_s, \hat{b}_s$ — current beliefs of vertex s

δ_h — scale factor of the coupling matrix

CONTENTS

CHAPTER

1

INTRODUCTION

1.1 Context

The massive amount of data produced in the so-called *Information Era*, in the first decade of the 21st century, is undoubtedly a very promising source of knowledge, but also a consolidated problem involving several issues such as storage, processing, and comprehension of the underlying phenomena. These issues are investigated by different research areas and pose a promising, yet, challenging problem. Such data is ubiquitously produced in our everyday life, from social network interactions and credit card transactions to digital breadcrumbs left by our Global Positioning System (GPS) systems. In 2010, a publication from *The economist* entitled *Data, data everywhere* (ECONOMIST, 2010) discussed the advantages, such as the potential of life and services improvements, and also the disadvantages, like privacy concerns that arise from this new availability of information. Finally, the publication emphasizes the importance of understanding and utilizing this data as a mean of pursuing several common interests of our society.

A significant part of this ubiquitously and constantly generated data come from the so-called web 2.0, in which users not only consume content but also generate content that is to be consumed by other users. This dynamic pushes data generation levels to a new ground, named Web-scale or planetary scale. Many times, this data can represent relationships between its elements, as in social networks, recommendation systems, online forums, electronic communication networks, citation networks, cybersecurity, and others. For that reason, it is intuitive and promising to model these systems as graphs, because it allows several properties to be explored, such as edge weights (representing for instance, the strength of a connection), edge and node data (such as a user's profile in a social network), and also the dynamic processes that take place in these networks (the diffusion of a hoax, for example).

1.2 Problem

Analyzing large scale networks, which can have up to billions of edges and nodes, requires a very efficient and powerful processing framework. Graphs like the YahooWeb (KANG *et al.*, 2011), Twitter (KRISHNAMURTHY; GILL; ARLITT, 2008) and Clickstreams (LIU; GUO; FALOUTSOS, 2009) are challenging due to their size, requiring large storage disks. It is not uncommon for these graphs to not fit in memory, which means that with inadequate processing it can take up to weeks until any analysis be obtained. One of the main strategies to process large graphs has been the use of distributed processing over computational clusters (LOW *et al.*, 2010) (KANG *et al.*, 2011) (MALEWICZ *et al.*, 2010), usually managed by frameworks like Hadoop (SHVACHKO *et al.*, 2010). This approach demands additional complexity, due to the distributed processing, as well as additional infrastructure cost, both of which can be prohibitive. Therefore it is interesting to be able to process large scale graphs in a single computational node, aiming at the definition of an analysis framework capable of discovering patterns, helping in the decision making process and comprehension, whilst maintaining a scalable, cheaper and simpler approach.

Following these considerations, new approaches that intend to circumvent scalability and complexity issues have been proposed in the years recent to this work. The more important are the ones that take advantage of the, now common, multi-core architectures, such as TurboGraph (HAN *et al.*, 2013), GraphChi (KYROLA; BLELLOCH; GUESTRIN, 2012), X-Stream (ROY; MIHAILOVIC; ZWAENEPOEL, 2013a), MMap (SABRIN *et al.*, 2013) and M-Flash (GUALDRON *et al.*, 2015). These frameworks employ techniques that allow a graph to be processed iteratively without the need of having the whole graph in memory, as well as allowing for several properties and algorithms to be calculated without the need for a complete traversal of the graph – which can sometimes be computationally inviable.

The aforementioned techniques refer to the use of the asynchronous parallel processing model, which can be oriented to edges or nodes; namely edge-centric and vertex-centric, respectively. That is, the processing is done iteratively, once for each edge, in the case of the edge-centric processing; or once for each node, in the case of the vertex-centric processing. Despite limiting the ways the network can be traversed, edge and vertex-centric algorithms are still very powerful, allowing for several tasks to be performed; specifically, discrete techniques that are based on matrix operations (matrix-matrix multiplications or matrix-vector multiplications, for instance) generalize under this paradigm. Examples of such algorithms include PageRank (PAGE *et al.*, 1999), spectral analysis (KANG *et al.*, 2014), diameter estimation (KANG *et al.*, 2008), connected components (ZHU; GHAHRAMANI, 2002) and random walks (KYROLA, 2013), as was shown by Kang, Tsourakakis and Faloutsos (2009).

1.3 Rationale for the choice of the research topic

In fact, due to the relevance of analyzing and comprehending different network properties, several works have been proposed in this context. Some of them analyze the temporal evolution of networks (AGGARWAL; SUBBIAN, 2014) (BRANDT; LESKOVEC, 2014) (MCAULEY; LESKOVEC, 2013); others focus on analyzing recommendations (GÜNNEMANN; GÜNNEMANN; FALOUTSOS, 2014) or mining sentiments (WEST *et al.*, 2014); some are focused on tensor decomposition (SIDIROPOULOS; PAPALEXAKIS; FALOUTSOS, 2014) (PAPALEXAKIS *et al.*, 2013) (SHIN; KANG, 2014) and enumerating triangles (PARK *et al.*, 2014). Another important consideration is security and quality of several online services (SHNEIDERMAN, 2015); in a mobile application store, for instance, it is important that user reviews remain legitimate because this factor can affect the credibility of the store (XIE; ZHU, 2014).

Furthermore, the need for efficient and cost-effective solutions can be perceived by a growing number of works that consider alternatives to the distributed processing approaches. In the work *CatchSync* (JIANG *et al.*, 2014), the authors define metrics to quantify the suspiciousness of a given user, Cristofaro *et al.* (2014) on the other hand, proposes an experiment where legitimate and illegitimate *likes* are compared, proposing metrics to help differentiate them. In another work, Shah *et al.* (2014) proposes an algorithm to detect suspicious behaviors, whereas Mao *et al.* (2014) recognizes malicious attempts in a governmental security network by using a multi-linear analysis algorithm.

Additionally, Gatterbauer *et al.* (2014) focuses on developing an efficient *belief propagation* algorithm to search for anomalies, such as fake profiles in social networks. Also Lin *et al.* (2014) proposes the use of a diffusion model to analyze the temporal evolution of a mobile application. Similarly, some works employ epidemiology concepts to model information diffusion (RODRIGUEZ *et al.*, 2014), while also trying to predict the size and duration of cascade-like effects in social networks (CHENG *et al.*, 2014). Finally, we also developed an algorithm that is capable of identifying potentially illegitimate promotion or defamation in online recommendation networks (GIMENES; CORDEIRO; RODRIGUES-JR, 2016).

1.4 Results and general hypothesis

With the results and discussions presented in this thesis, we consider that we were able to develop processing techniques for large-scale networks, building towards analyzing some of the underlying properties that emerge from these complex networks in different domains. As our main contributions we managed to:

- investigate algorithms and properties of networks, their comprehension and requirements for interpretation in specific domains, as shown in Chapter 3;

- work with a large scale graph processing framework, using vertex-centric asynchronous parallel processing to enable efficient computation in a single computational node, as shown in both Chapters 3 and 5;
- finish the project that started during the MSc, culminating in the publication ORFEL (GIMENES; CORDEIRO; RODRIGUES-JR, 2016) that focused on fraud detection in large scale recommendation systems; also presented in Chapter 3;
- propose our novel vertex-centric belief propagation algorithm *VCBP* that improved the state-of-the-art by enhancing the applicability and efficiency such an algorithm, detailed in Chapters 2, 4 and 5

Given these contributions, this doctorate thesis can be expressed through the following general hypothesis, which this thesis shall find valid or invalid based on the standard scientific protocol:

General Hypothesis: *The use of iterative vertex-centric asynchronous parallel processing techniques over multi-core computational architectures can lead to the development of efficient algorithms capable of revealing interesting patterns in large scale networks over real-world domains, such as social networks, recommendation systems, and citation networks.*

1.5 Document organization

This document is structured in a way that we start as broad as possible whilst gradually narrowing the scope towards the more specific results of this thesis. The Chapters are organized as follows:

Chapter 2 explains and reviews the literature related to graph processing techniques and paradigms.

Chapter 3 reviews an extensive scope of concepts related to graphs/complex networks while simultaneously presenting practical applications, methodologies, results and interpretations revolving around those concepts in multiple domains comprising three of our published works.

Chapter 4 discusses the history and characteristics of the Belief Propagation algorithm.

Chapter 5 details our methodology and the results for our proposed *VCBP* algorithm and this thesis main contribution.

Chapter 6 closes this thesis presenting conclusions and future works.

CHAPTER

2

VERTEX-CENTRIC ASYNCHRONOUS PROCESSING

2.1 Initial considerations

This chapter presents one of the bases of this thesis: the vertex-centric graph processing paradigm, including a review of the more broad graph processing literature as well as specific works that inspired and helped the development of our work. Although not intended as an exhaustive list of all the frameworks and techniques, we focus on seminal works that originated schools of thought when it comes to processing graphs, allowing the reader to understand the overall state of the art and different approaches to be used.

2.2 Overview

Graph processing, in general, has been transitioning from a more traditional whole-graph available model to a more decentralized design. That is, conventional graph algorithms such as Dijkstra's shortest path receive the whole graph as input and considers that the data is available in memory, making frequent use of random accesses. However, with the unprecedented increase in the size of real-world graphs and data applications, newer and more scalable approaches have been and are still being proposed to address such a problem. One of the more commonly proposed methodologies focuses on partitioning the data, and using distributed frameworks such as MapReduce (DEAN; GHEMAWAT, 2004) to process the graphs.

Yet, due to the high inter-dependency between graph partitions, these generic approaches fail to leverage specific graph information to optimize the processing; hence, several of such attempts failed to produce the desired performance, which led to more specific research and development regarding large-graph processing. We can divide these proposals into two processing categories: Distributed and Single-Machine.

2.3 Distributed graph processing

Distributed systems take advantage of large computational clusters to divide-and-conquer, by splitting the graph into smaller partitions and processing each part in a different node. Several works follow this route such as: Pegasus (KANG; TSOURAKAKIS; FALOUTSOS, 2009), GraphLab (LOW *et al.*, 2012), Pregel (MALEWICZ *et al.*, 2010) and PowerGraph (GONZALEZ *et al.*, 2012). The issues of this approach are the high cost associated with the infrastructure, the complexity associated with the programming model, and the graph partitioning scheme; which can lead to heavy network communication and unnecessary waste of processing power.

More specifically, GraphLab (LOW *et al.*, 2012) is an asynchronous parallel system that relies on a shared-memory abstraction called GAS (Gather, Apply, Scatter); in this model, a vertex can discover information about its neighbors in the Gather phase, effectively perform the algorithm computations in the Apply phase, and finally update its adjacent edges and vertices during the Scatter phase. In order to maintain serializability during asynchronous processing, GraphLab utilizes a neighbor-locking protocol. Similarly, PowerGraph (GONZALEZ *et al.*, 2012) builds upon these concepts by proposing an architecture where high-degree nodes are processed by more than one worker, in order to avoid imbalance issues on power-law graphs.

Pegasus (KANG; TSOURAKAKIS; FALOUTSOS, 2009) is one of the frameworks that build upon the MapReduce (DEAN; GHEMAWAT, 2004) technology via proposing a generalized iterative matrix-vector multiplication model called GIM-V; one of the first works to propose this model, Pegasus suffers from the necessity of synchronizing the whole graph-state between iterations, even when most of the graph remained unchanged, such overhead leads to performance inefficiencies.

Finally, Pregel (MALEWICZ *et al.*, 2010) is one of the main precursors to the BSP (Bulk Synchronous Parallel) programming model, using a message passing processing model to address the communication between multiple workers and data segments. Additionally, in such a model it is possible to declare one or more vertices inactive during any iterative step. Leveraging the TLAV (Think Like a Vertex) programming framework - which is discussed in more details further ahead in subsection 2.4.1 - Pregel facilitates the user programming costs by allowing the user to focus only on the computation that happens inside a vertex while not needing to worry about the underlying graph representation structure.

Other works (TAMERSOY; ROUNDY; CHAU, 2014) (PANDIT *et al.*, 2007) (MCGLOHON *et al.*, 2009) fail to consider large-scale scenarios, whereas others make use of clusters (KANG; CHAU; FALOUTSOS, 2011) or supercomputers (CHAU *et al.*, 2011) to achieve scalability.

2.4 Single-machine large graph processing

Alternatively to the distributed approach, in 2012, scientists started proposing single-machine frameworks that rely on iterative parallel processing and efficient disk-based techniques. These include both Vertex-Centric and Edge-Centric processing frameworks, such as Graphchi (KYROLA; BLELLOCH; GUESTRIN, 2012), TurboGraph (HAN *et al.*, 2013), X-Stream (ROY; MIHAILOVIC; ZWAENEPOEL, 2013b) and MFlash (GUALDRON *et al.*, 2015).

Aiming to explore multi-core architectures and disk-based I/O optimizations to provide scalability over cheaper machinery, these approaches also avoid the distributed pitfalls of needing to partition the graph over several computational nodes. Reducing the communication overhead and being able to more specifically optimize the processing of the graphs based on underlying structural properties and characteristics. Additionally, due to the lower complexity of the overall system, programming algorithms for single-machine graph processing frameworks tend to be simpler and more straight-forward than their distributed counterparts.

Graphchi (KYROLA; BLELLOCH; GUESTRIN, 2012) is a vertex-oriented graph processing framework that implements the Parallel Sliding Windows (PSW) mechanism, which is a disk-based graph access model that focuses on minimizing the number of random accesses needed to perform a given task. PSW works by partitioning the graph into *shards*, which allows large graphs that do not fit in the main memory to be sequentially loaded via these partitions. Each partition is an ordered subgraph; edges are stored based on the source vertex ID. Graphchi also allows for selective scheduling of vertices, avoiding unnecessary processing of already stabilized vertices in every iteration. TurboGraph (HAN *et al.*, 2013) has a similar approach to Graphchi, while focusing mainly on the advantages in the performance that can be obtained when using modern Solid State Drives (SSD).

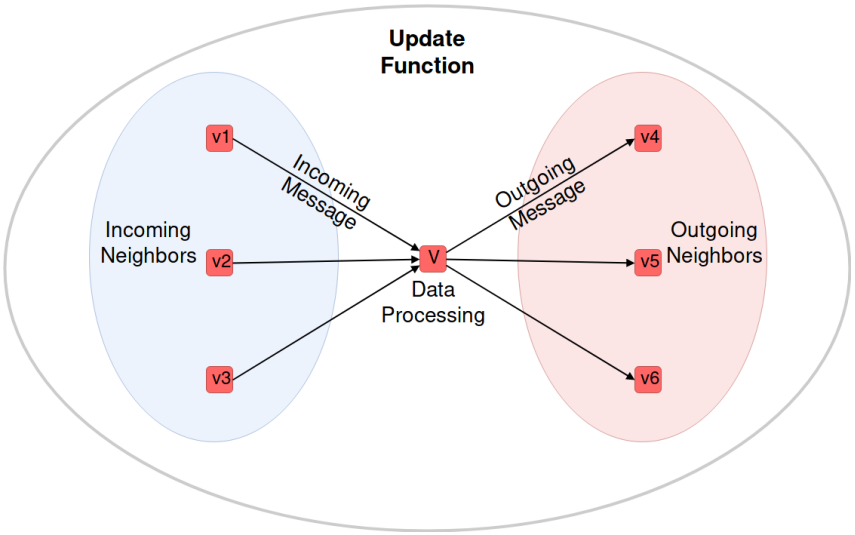
X-Stream (ROY; MIHAILOVIC; ZWAENEPOEL, 2013b), on the other hand, is an edge-centric framework that proposes an unordered edge list stream processing technique. X-Stream manages to lower the preprocessing costs by eliminating the need to order the edges before starting to process the graph. Whilst MFlash (GUALDRON *et al.*, 2015) proposes a different block-based graph partitioning system to reduce disk-accesses that focus on fully exploiting the capabilities of modern hard drives.

As mentioned before, there are two main approaches to single-machine large graph processing frameworks, Vertex-Centric and Edge-Centric, and below we detail each:

2.4.1 Vertex-centric graph processing

Vertex-centric, vertex-oriented or TLAV algorithms are computations oriented to the standpoint of each vertex in the graph. That is, the user-defined computation is iteratively processed over vertices of the graph, considering only localized information for the computation such as the vertex's own information as well as its incoming-edges messages. Vertex-centric

Figure 1 – Update function representation.



Source: Elaborated by the author.

programs can be executed both synchronously where each vertex computes its update function once and then waits for every vertex to finish before starting the next iteration, as well as asynchronously, when a vertex finishes its processing it can immediately begin the next iteration. Several methods exist to control the execution of the algorithm including local and global barriers, synchronization locks, and scheduling techniques depending on the implemented framework.

Due to the local nature of the processing, vertex-centric techniques cannot express every graph algorithm, particularly it has difficulties implementing algorithms that require a graph-omniscient approach. However, such a processing model can still perform several common and useful graph algorithms, such as the PageRank (PAGE *et al.*, 1999) algorithm that calculates the importance of web-pages. PageRank is a good example of a problem that can be solved both by traditional shared-memory sequential processing models as well as using the iterative parallel vertex-centric approach. By leveraging the inherent parallelism available to the vertex-centric approach, it is possible to process the PageRank algorithm efficiently, enhancing the scalability of the algorithm when compared to traditional methods.

At its core, in a vertex-centric graph processing system, each vertex has associated values and iteratively updates its values based on incoming messages from in-edges and then propagates such changes to its neighbors via out-edges. Figure 1 shows the basic structure of vertex-centric models, the *Update Function*. This approach is well suited to work with large graphs in a single computational node due to the inherent parallelism and asynchronicity allowing for efficient usage of computational resources such as the multiple cores available in modern processors, as well as the higher rates of data transfer in SSDs.

The vertex-centric graph processing is based on iteratively executing the update function for each scheduled vertex in the graph until the algorithm converges. Algorithm 1 presents an

overview of the model. The first part of the algorithm represents the overall application loop, that is we iteratively process every node in the graph by calling its update function. In the second part of the algorithm, the actual update function is presented. Each vertex has access to its own information as well as incoming and outgoing edges; this differs from omniscient graph processing paradigms where the whole graph is available in memory at any point in the computation.

Therefore computation inside an update function usually happens as a variation of the presented pseudo-code - first the function accesses neighboring information, then performs local computation according to the algorithm being implemented, and finally broadcasts the new information processed so that adjacent vertices may access it. In this step, we can discuss the difference between synchronous and asynchronous execution of the algorithm. If we consider synchronous execution while the vertices may update their values and interfere with adjacent edges that information is only going to be consolidated when the iteration ends, therefore neighboring vertices will not have access to the new values until the next update function. Whereas if we are running the algorithm asynchronously all of the newly update values are immediately made available to neighboring vertices - this means that effectively information propagates inside the same iteration also. This is controlled internally via scheduling techniques, limiting which vertices are processed in each iteration as well as inside the same iteration. Leveraging the potential asynchronous capabilities of the model can result in more efficient programs as shown and discussed in detail in Chapter 5. On the other hand, it is important to note that not all algorithms and computational problems can be processed asynchronously as several problems depend upon the synchronization steps to guarantee the correct execution of the program.

Algorithm 1 – Vertex-Centric Algorithm Structure

```
1: procedure VC-PROGRAM(Graph G)          ▷ The program receives the input graph
2:     while Convergence is not achieved do
3:         for each Vertex v ∈ G do
4:             Update(v)
5:         end for
6:     end while
7: end procedure
   procedure UPDATE(Vertex v)             ▷ The update function receives the vertex
2:     x[] <- read values of incoming edges of v
       v.value <- f(x[])                  ▷ Perform computations and update v
4:     for each Edge e ∈ v do
           e.value <- g(v.value, e.value) ▷ Propagates the changes to neighbors
6:     end for
   end procedure
```

Another important part of the vertex-centric model is related to its iterative nature. Depending on the algorithm it may be more interesting to run the program for a fixed number of

iteration. However most of the time we will be interested in running the algorithm as efficiently as possible and that means stopping as soon as the computation has finished or more specifically as soon as the algorithm reaches convergence. Convergence can vary from program to program, but in general, it refers to a stopping point in the algorithm when there is no need to continue running it given that it would not significantly improve the results. Most of the time we are interested in making sure that our output is somewhat stable between iteration so that we can finish the execution. This can be done both in absolute terms - the execution ends when the output change from one iteration to the next is less than a ε; as well as in relative terms - the execution stops when the ratio between the output of one iteration and the next is less than a ε.

In order to exemplify the usability of a vertex-centric program, Algorithm 2 shows how one could implement the classic PageRank algorithm as previously presented in Section 3.2.2. In the algorithm we first loop through the incoming edges of the vertex, computing the summation of the neighbors ranking weighted by the edge value. We then update the current vertex value based on the calculated sum, and finally broadcast the change to outgoing edges by updating their value to the new vertex value. The update-function is executed for every vertex in the graph iteratively for a given number of iterations following the general structure already presented in the first part of Algorithm 1.

Algorithm 2 – Update-function for the PageRank algorithm.

```
1:  procedure UPDATE(Vertex v)
2:      var sum <- 0
3:      for each Edge e ∈ v.InEdges() do   ▷ Loops through in-edges computing sum of ranks
4:          sum += e.weight * (e.weight * e.neighborRank)
5:      end for
6:      v.value <- 0.15 + 0.85 * sum
7:      for each Edge e ∈ v.outEdges() do                ▷ Broadcasts new rank to out-edges
8:          e.neighborRank <- v.value
9:      end for
10: end procedure
```

Source: Adapted from Kyrola, Blelloch and Guestrin (2012).

This example shows how intuitive it can be to express iterative algorithms in the vertex-centric paradigm, providing not only the already discussed benefits in terms of scalability but also simplifying the programming overhead necessary to create efficient parallel algorithms.

2.4.2 Edge-centric graph processing

Initially proposed in X-Stream (ROY; MIHAILOVIC; ZWAENEPOEL, 2013b), edge-centric graph processing is similar to vertex-centric approaches in the sense that it also relies on local and iterative processing. In edge-centric models, the graph is considered a stream of edges and computation is performed in two steps, as shown in Algorithm 3, which presents the

processing model. In the first step, all the edges are processed distributing the source vertex values and creating a list of the necessary updates. In the second step, every update is processed, this time performing the computation and updating of the values.

Algorithm 3 – Edge-Centric Algorithm Structure

```
1:  procedure VC-PROGRAM(Graph G)          ▷ The program receives the input graph
2:      while Convergence is not achieved do
3:          for each Edge e ∈ G do
4:              Scatter(e)
5:          end for
6:          for each Update u ∈ G do
7:              Update(u)
8:          end for
9:      end while
10: end procedure
    procedure UPDATE(Update u)
2:      Applies u to the destination vertex
    end procedure
    procedure SCATTER(Edge e)
        Sends needed updates through e
3:  end procedure
```

One of the characteristics of the edge-centric model proposed by Roy, Mihailovic and Zwaenepoel (2013b) is that it iterates over an unordered edge list, eliminating the need to sort the graph during preprocessing. However, due to the nature of the two-step processing model, it can also lead to performance loss due to having to stream the entire edge list in order to perform computation in a given edge. Whilst vertex-centric programs can also suffer from this same issue the single-step nature of vertex-centric algorithms helps to mitigate this issue. Additionally, while it is possible to express the same algorithms in both paradigms - because they are intrinsically very similar - we find that it is easier to understand and develop algorithms for the traditional vertex-centric framework from the programmer's standpoint.

2.5 Our choice

When considering the available graph processing frameworks to develop our work we chose the vertex-centric approach, particularly the GraphChi framework (KYROLA; BLELLOCH; GUESTRIN, 2012). We analyzed different options and while there are multiple frameworks in the literature to choose from, Graphchi remains competitive while also being well-documented, having great programmability and offering out-of-the-gate asynchronous parallel processing (CAPELLI *et al.*, 2019). Additionally, GraphChi also provides built-in in-memory execution for smaller graphs that fit in memory seamlessly, making it so that its practical use is unrivaled.

More specifically, in this work, we departed from the summation-based version of BP, based on which we designed a vertex-centric version of the message-passing principle. Following the vertex-centric model, we engineered an algorithm that runs on asynchronous parallel processing frameworks. Our design breaks the computation in asynchronous parallel steps, taking full advantage of the intrinsic parallelism of the algorithm. As a result, it significantly improves the speed and the convergence ratio when compared to previous algorithms. As our objectives included: an algorithm that can be expressed over the paradigm; the ability to process large graphs that may not fit in main memory; as well as the necessity of asynchronous capabilities, GraphChi proved to be the correct choice.

2.6 Final considerations

This chapter discussed the processing paradigm options and compared several approaches. We also present the reasoning behind our choice of framework and processing technique.

CHAPTER

3

APPLIED CONCEPTS

3.1 Initial considerations

This chapter presents and explores the general concepts related to graph processing and characterization. The objective is to introduce the building blocks that were important to the doctoral thesis. We observe that, beyond a simple exposition of definitions, we demonstrate the use of relevant theories over real and synthetic datasets as detailed in the sections related to each of our published works. Such works include ORFEL (GIMENES; CORDEIRO; RODRIGUES-JR, 2016), which is the culmination of research started during my MSc degree, and expanded during the Phd; it focuses on fraud detection and scalability for large graph processing. We also discuss two collaborative works (SPADON; GIMENES; RODRIGUES-JR, 2017)(SPADON; GIMENES; RODRIGUES-JR, 2018) that focus on overlapping research areas between a colleague *Gabriel Spadon* and myself, focusing on applied complex network analysis, specifically for the domain of street-networks. The goal is to demonstrate the framework composed of data, programming libraries, tools, and code used and developed during the project as well as presenting the additional contributions derived from our work.

3.2 Complex network analysis

Science has always tried to understand the underlying phenomena in all areas of human knowledge. With the advent of powerful computers, many seemingly impossible tasks began to be solved by simulating potentially costly and long experiments in the blink of an eye. Recently, scientists have started analyzing and understanding how complex systems work with the help of graph theory. That is, by using the graph model to represent a system's complex structure, it is possible to emulate its emergent behavior, something that was not possible by analyzing its parts separately.

Such a course of action can help to understand and solve several important and current

Table 1 – Table of symbols

$G(V,E)$	G is a network where V is the set of vertices and E is the set of edges
$\Gamma(x)$	set of neighbors of vertex x
$e(x,y)$	edge between vertices x and y
$d(x,y)$	shortest path distance between x and y

problems of society. Examples include analyzing how epidemic viruses spread using, for instance, airline networks; detecting suspect behavior in online recommendations, to prevent customers from being scammed; understanding how proteins interact to discover their importance; mapping brain functions to treat degenerative diseases, and several other applications. Therefore, in this chapter, we present applied concepts related to the multitude of domains we worked with.

3.2.1 Global measures

In this section, we present global features useful in the understanding of the work to be presented in this document. Along the text, please refer to Table 1 for the symbols used throughout this chapter.

Degree distribution

Arguably, the most important and defining characteristic of a network is its degree distribution. Several properties can be obtained directly from it, and several others can be derived. There are known distributions observed in complex networks, such as the uniform distribution that comes from Random Graphs. But the most common distribution in the real world, that is, both in nature as well as in human environments is the Power-law distribution. Networks that follow this distribution are known as scale-free networks. The degree distribution in scale-free networks is given by:

$$P(k) \propto k^{-\gamma}$$

where $P(k)$ stands for the fraction of nodes that have degree k in the network and γ is a constant, known as the scaling factor.

In order to empirically find the best γ and k_{min}, that is the minimum degree for which the distribution starts to follow a power-law, one need to iteratively compute the following expression for each k_{min} while using a distance measure to define the best value:

$$\gamma = 1 + \frac{n}{\sum_{i=1}^{n} ln(\frac{k_i}{k_{min}})}$$

Mean degree

One of the more common network measures is the vertex degree, that is, the number of connections that a given vertex has. As for the network itself, one can consider the average degree of all vertices:

$$\langle K \rangle = \frac{1}{|V|} \sum_{v \in V} |\Gamma(x)|$$

Second moment of the degree distribution

The second moment of the degree distribution is the variance, intuitively representing the disparity between the degree of different nodes in the network, defined by:

$$\langle K^2 \rangle = \frac{1}{|V|} \sum_{v \in V} P(v)(k_v - \langle K \rangle)^2$$

an example of the usage of this measure is that it can be be used to calculate the theoretical activation threshold for an epidemic spread.

Shannon entropy of the degree distribution

Entropy is a widely used concept in areas such as information theory, and intuitively represents the amount of uncertainty in the given information. In the context of complex networks, it can be used to quantify, for instance, the network's resistance to attacks. The entropy of a network is given by:

$$\mathscr{H}(G) = -\sum_{k} P(k) \log P(k)$$

where $P(k)$ is the fraction of vertices in the network that have degree k. It can also be interpreted as being the probability of a random vertex having exactly degree k.

Local clustering coefficient

The local clustering coefficient measures how sparse (or connected) is the neighborhood of a given vertex.

$$LCC(x) = \frac{2\,|E_x|}{(|\Gamma(x)|)(|\Gamma(x)| - 1)}$$

where $|E_x|$ is the number of edges between neighbours of vertex x. If we were considering a directed network, we would have $|E_x|$ instead of $2\,|E_x|$.

Mean local clustering coefficient

As the name suggests, the mean local clustering coefficient is the average local clustering coefficient throughout the whole network. That is:

$$\mathscr{C}(G) = \frac{\sum_{v \in V} LCC(v)}{|V|}$$

which gives a quantitative idea about the connectivity of the local neighborhood in the network.

Transitivity

Similar to the MLCC, transitivity measures how sparse or connected is a network, with the difference of giving all triangles in the network the same weight, while MLCC weighs vertices equally. We can define transitivity as:

$$\mathscr{T}(G) = \frac{3N_{\triangle}}{N_3}$$

where $N_{\triangle}$ is the number of triangles in the network and N_3 is the number of connected triplets.

Average shortest path

One might be interested in evaluating how quickly can information travels through a network, and one of the basic properties related to that task is finding what the average shortest path is, that is, measuring the average travel distance between all pairs of nodes. That can be done via:

$$\mathscr{L}(G) = \frac{1}{(|V|)^2} \sum_{x \in V} \sum_{y \in V} d(x,y)$$

Efficiency

Efficiency measures how efficient is the communication in a network, that is, how fast and reliable is the propagation of information. This is done by calculating:

$$\mathscr{E}(G) = \frac{1}{(|V|)((|V|)-1)} \sum_{x \neq y} \frac{1}{d(x,y)}$$

Diameter

The diameter of a network is the largest shortest path distance between all pairs of nodes. Given by:

$$\mathscr{D}(G) = \max_{x,y \in V} d(x,y)$$

3.2.2 Centrality

Another important set of measures are the centrality-related measures, which can be used to determine the most interesting vertice depending on the desired characteristics. Such a concept can be used in real-world problems like finding the best address to allocate a hospital or deciding how to work marketing campaigns.

Degree centrality

One of the more common network measures is the vertex degree, that is, the number of connections that a given vertex has. This measure can also be used in a centrality context, that is, the bigger the degree of the vertex, the more central it can be considered.

$$K(x) = |\Gamma(x)|$$

Betweeness centrality

Betweenness, proposed by Freeman (1977) accounts for the importance of a node when considering all shortest paths in a network. That is, by determining the fraction of all the shortest paths that happen to pass through a given vertex. Intuitively, higher values of betweenness indicate that the vertex is part of more paths, and therefore can be considered more central. The measure is calculated as follows:

$$B(x) = \sum_{s \neq x \neq t \in G} \frac{\sigma_{st}(x)}{\sigma_{st}}$$

where $\sigma_{st}(x)$ corresponds to the number of shortest paths between s and t, that go through x; and σ_{st} accounts for all of the shortest paths between s and t.

Eigenvector centrality

Another centrality measure is the Eigenvector centrality, which tries to compute the centrality of a node based on the centrality of its neighbors. It can be written in its matrix form or solved iteratively. In its matrix formulation, we can write:

$$Ar = \lambda r$$

where A is the adjacency matrix of the graph and λ is the largest eigenvalue associated with the eigenvector of matrix A (NEWMAN, 2010).

Closeness centrality

Closeness centrality, as proposed by Freeman (1979) considers the inverse of the sum of all shortest paths starting on the given vertex and ending on every other vertex on the network. That is, it computes the distance between a vertex and the rest of the network. which intuitively should relate to how central a vertex is, the closer he is to the rest of the network. That being so, we can write:

$$C(x) = \frac{1}{\sum_{y \neq x \in G} d(x,y)}$$

Page Rank

Following the same idea of the Eigenvector centrality, the Page Rank metric also tries to gauge a vertex centrality based on its neighbors. Additionally, PageRank weighs each neighbor's contribution according to the number of edges it has, therefore considering also the exclusivity of the link. Finally, it also incorporated a random factor, which accounts for the probability of a random node being visited disregarding existing nodes, which makes sense in the context it was proposed, that is, to rank the results of the Google search engine. Proposed by Page *et al.* (1999), we can write Page Rank as follows:

$$PR(x) = \frac{1-d}{|G|} + d \sum_{\forall y | x \in \Gamma(y)} \frac{PR(y)}{|\Gamma(y)|}$$

where d is the random factor and $|G|$ is the number of nodes in the network.

Assortativity

Assortativity, as defined by Newman (2003), is a measure that models the similarity of connections in the network with respect to the node's degree. That is, it tries to understand if the nodes tend to connect with other nodes that have similar degrees, or if nodes with a higher degree tend to connect with nodes with a lower degree, or even if there is no correlation between the degrees of neighbors.

3.2.3 Community detection

One of the more prominent research areas in network analysis is the community detection area, born from the same basic concepts as the clustering data mining area. Unfortunately, one cannot simply use the same algorithms and theoretical solutions for complex networks. Therefore there is a constant flux of new algorithms, techniques, and concepts related to community detection in the literature.

By analyzing the community structure in a network one can better understand how different groups of vertices interact as well as understand where does a given vertice fits in terms of the overall network structure. Practical examples of applications include detecting frauds, grouping customers based on their profiles, optimizing network routing and several others.

Therefore, we focus on whether or not vertices can be grouped according to their connections. That is, can we find sets of vertices that have significantly more connections between them as opposed to connections with other vertices in the network?

Additionally, if we are indeed able to find such communities, this is strong evidence that nodes within the same community must share common properties, as stated by Fortunato (2010). Communities also have several applications such as identifying customers with similar interests and classifying nodes according to their memberships. To gauge how well does a partition divide the network into communities, one can use the Q modularity measure, as defined by Newman (2004):

$$Q = \sum_i (e_{ii} - a_i^2)$$

where we compute the fraction of edges that fall within the same communities, minus the expected fraction of edges that would appear between communities if these edges had been randomly assigned.

Finally, many algorithms and techniques were proposed for the specific task of finding the community structure in complex networks, including the ones we analyze in this section: Betweenness-centrality based, Fast greedy, Walktrap and Leading Eigenvector.

Betweeness centrality based

The first method we analyze was proposed by Girvan and Newman (2002) and consists of iteratively removing the edge with the highest betweenness centrality until there are no more edges left. This is considered a divisive algorithm, as one starts with the whole graph and ends up with each edge in its own community. The authors show that the method works for both computer-generated data as well as real-world datasets with a satisfactory success ratio. Nonetheless, it is important to note that this algorithm is fairly intensive in terms of computational complexity, given that we need to calculate the betweenness for every edge multiple times.

Fast Greedy

The Fast Greedy algorithm is an agglomerative algorithm that works by joining pairs of vertices while trying to maximize the modularity measure directly. Proposed by Clauset, Newman and Moore (2004), one of the main advantages of the algorithm is, as the name suggests, the computational efficiency of the algorithm, arguably running in linear time for real-world graphs.

Walktrap

Walktrap is also an agglomerative algorithm, proposed Pons and Latapy (2006), it leverages random walks to calculate the distance between two communities, merging the closest ones according to the mean of the squared distances between each vertex and its community. It is also computationally efficient, as the authors propose several resourceful techniques to update the distances iteratively.

Leading eivenvector

Finally, the leading eigenvector method (NEWMAN, 2006) is a divisive algorithm based on the concept of a modularity matrix. By leveraging the known properties of random networks (specifically the ones generated via the configuration model), the authors propose to divide the graph based on the discrepancy of probabilities between the real network and the corresponding random graph. This is done by calculating the largest eigenvector for the modularity matrix, similarly to how the Laplacian matrix is used in graph partitioning techniques.

3.2.4 Multidimensional projection

In our work to characterize street-networks (SPADON; GIMENES; RODRIGUES-JR, 2018), we consider multidimensional projection as a dimensionality reduction tool, representing higher dimensional data in an embedded lower-dimensional space. We use both a non-linear algorithm called Isomap and a linear alternative called PCA.

Isomap (TENENBAUM; SILVA; LANGFORD, 2000) combines ideas from the original Floyd-Warshall algorithm with classic multidimensional scaling by computing positions for each pair of vertices based on their distances, and therefore effectively estimating the full pair-wise matrix from which the algorithm computes the reduced-dimensional points. PCA (RINGNÉR, 2008) on the other hand uses an orthogonal transformation to convert a set of observations into a set of linearly uncorrelated points.

3.2.5 Cluster analysis

Separating objects into groups following the general idea that objects that find themselves within the same group are considered more similar to each other than to objects that are in different groups is an intuitive task, often referred to as clustering or cluster analysis. Several algorithms and domain-specific techniques exist, the concept is similar to the already discussed community detection problem 3.2.3.

In our work ORFEL (GIMENES; CORDEIRO; RODRIGUES-JR, 2016) we use the concept of co-clustering or bi-clustering in the context of detecting lockstep behavioral patterns, as is shown in detail in section 3.3. Whereas when grouping similar cities (SPADON; GIMENES;

RODRIGUES-JR, 2018) we use the widely regarded KMeans (LLOYD, 1982) algorithm to split our data into groups of equal variance, minimizing the distance within the clusters.

In addition to clustering algorithms, one common concept related to grouping techniques is the validation of the partitions. There are several metrics dedicated to measuring partitioning quality. Including the ones we employed in our work: the Silhouette score (PAKHIRA; BANDYOPADHYAY; MAULIK, 2004) and the Dunn index (PAKHIRA; BANDYOPADHYAY; MAULIK, 2004) both of which fall into the internal-quality metrics classification, meaning that they do not require prior knowledge about the dataset.

3.3 ORFEL - Detection of defamation or illegitimate promotion in online recommendation

This Section depicts the problem, development, and results achieved in our work ORFEL (GIMENES; CORDEIRO; RODRIGUES-JR, 2016). ORFEL is a fraud detection algorithm that focuses on finding systematic attacks in recommendation systems, that is, trying to differentiate legitimate reviews from ill-intended interactions. Our work also considers the ever-increasing problem of scalability, working with web 2.0 systems that can have very large user bases and therefore a large amount of recommendation data to parse and investigate, by providing a scalable framework capable of processing large-graphs efficiently and cost-effectively.

3.3.1 Problem and method

Fraud detection

At its core, ORFEL identifies lockstep behavior in recommendation system bipartite graphs of users and products. Known in the literature as co-clustering or bi-clustering we aim to partition both the rows and columns of the adjacency matrix, in this case, users and products. Other works have addressed the bi-clustering problem such as Papalexakis and Sidiropoulos (2011) and Papalexakis, Beutel and Steenkiste (2012), who used PARAFAC decomposition techniques, and Dhillon, Mallela and Modha (2003) that worked with information theory methods over text documents.

In 2012, Douceur (2002) managed to propose one of the first algorithms that tried to find fraudulent interactions on the web, from this work the term Sybil attacks where a single entity impersonates multiple identities to interfere with the normal functioning of the system. Other types of attacks were analyzed, such as the shilling attack; in which fake users are used to undermine the reliability of recommender systems. However, to be able to accurately detect attacks in the domains we are interested, it is also important to consider the interaction timestamp, making the problem more complex (NP-Hard), which prevents more ambitious goals of finding the best solution. The CopyCatch algorithm (BEUTEL *et al.*, 2013) also takes into consideration

the timestamps but leverages distributed technologies to achieve satisfactory performance whilst not considering any weight in the graph's edges, that is, it cannot take into consideration ratings, such as the 1-to-5-star commonly used rating system.

Lockstep formulation

More formally, ORFEL is interested in finding sub-sets that can be expressed by the following definition:

Definição 1. A set of products P and a set of users U comprise an $[n, m, \Delta t, \rho]$-*temporally-coherent near bipartite core* if and only if there exists $P_i \subset P$ for all $i \in U$ such that:

$$|P| \geq m \tag{3.1}$$

$$|U| \geq n \tag{3.2}$$

$$|P_i| \geq \rho |P| \ \forall i \in U \tag{3.3}$$

$$(i, j) \in E \ \forall i \in U, j \in P_i \tag{3.4}$$

$$\exists t_j \in \mathbb{R} \ s.t. \ |t_j - L_{i,j}| \leq \Delta t \ \forall i \in U, j \in P_i \tag{3.5}$$

Or simply, we are interested in a set of products P, that was recommended by a set of users U, within a δt time-window. Parameter ρ is responsible for relaxing the definition when we also want those groups that follow the definition within a certain percentage. Figure 2 illustrates the aforementioned definition, in the figure a group of users (1,2 and 3) recommends a group of products (A and B), within limited time-windows for each product, forming a bipartite core. This modular definition of lockstep is also interesting because it is possible to tailor each parameter according to domain-specific real-world constraints and prior knowledge, allowing the algorithm to adapt to several different scenarios and applications.

Finally, it is then possible to formalize the task as an optimization problem; and taking into consideration the weights of the edges - that is, the system's specific rating score model - we can further define the concepts of illegitimate promotion and defamation. Illegitimate promotion is characterized by weight scores larger than κ, whereas defamation corresponds to weight values smaller than κ, where κ represents the average score, that is, the closest to a neutral recommendation. Table 2 presents the definitions needed for our problem formalization.

Given that our objective is to catch as many suspect users as possible, while only growing P until parameter m is satisfied we can define our objective function as in Equation 3.6. The goal is then to find $U[c]$ and $P[c]$ to maximize the number of users and their interactions for a given cluster c.

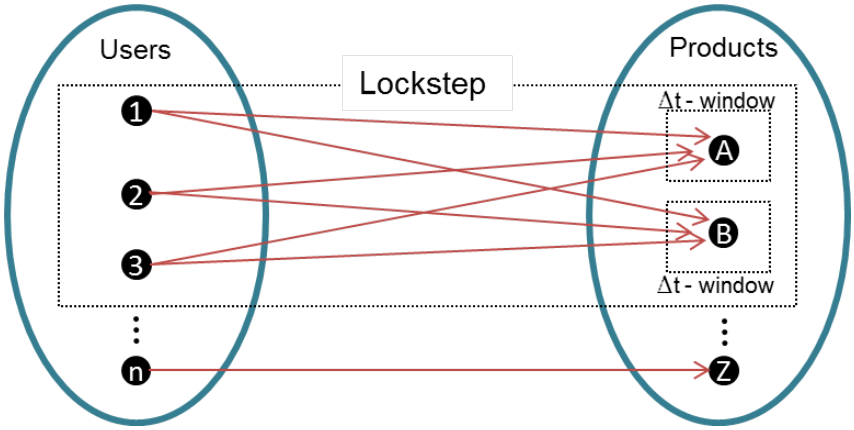


Figure 2 – Lockstep illustration.

Source: Gimenes, Cordeiro and Rodrigues-Jr (2016).

Symbol	Definition
M and N	Number of nodes in each side of the bipartite graph.
C	Set of locksteps.
I	$M \times N$ adjacency matrix.
L	$M \times N$ matrix holding the timestamp of each edge.
W	$M \times N$ matrix holding the weight of each edge.
$U[c]$ and $P[c]$	Set of users or products in lockstep c.
m and n	Minimum number of products and users in the lockstep to be considered valid.
Δt	Size of the timespan.
ρ	Threshold percentage that the cardinality of the sets of products and users must satisfy to be in a lockstep.
nSeed	Number of starting seeds for the algorithm to begin searching for locksteps.
λ and κ	Function and threshold used to define defamation and promotion.
v_j	Current average time of suspicious recommendations to product j.

Table 2 – Symbols and Definitions.

Source: Gimenes, Cordeiro and Rodrigues-Jr (2016).

$$\arg\max_{U[c],P[c]} \sum_{i} q(L_{i,*}|c, W_{i,*}|c, P[c]) \tag{3.6}$$

where

$$q(u,w,P[c]) = \begin{cases} \sigma \text{ if } \sigma = \sum_{j \in P[c]} I_{i,j} \phi(v_j, u_j) \lambda(w_j) \geq \rho m \\ 0 \text{ otherwise} \end{cases} \tag{3.7}$$

$$\phi(t_v, t_u) = \begin{cases} 1 \text{ if } |t_v - t_u| \leq \Delta t \\ 0 \text{ otherwise} \end{cases} \tag{3.8}$$

$$\lambda(g_j) = \begin{cases} 1 \text{ if } g_j \geq \kappa \\ 0 \text{ otherwise} \end{cases} \quad \text{for promotion} \tag{3.9}$$

$$\lambda(g_j) = \begin{cases} 1 \text{ if } g_j \leq \kappa \\ 0 \text{ otherwise} \end{cases} \quad \text{for defamation} \tag{3.10}$$

Equations 3.9 and 3.10 correspond to our concepts of illegitimate promotion and defamation, respectively, while Equation 3.7 shows how we incorporate these weight constraints in the overall formulation, through the definition of a threshold λ.

ORFEL has five parameters: m, n, ρ, Δt and *nSeeds*. The first two, m and n, respectively refer to the minimal amount of *products* and *users* that the algorithm searches for when considering suspicious locksteps. Parameter ρ is the minimum percentage (the tolerance fraction) of products $\rho * m$. Parameter Δt sets the desired time-window. And *nSeeds* refers to the number of seeds that are spread on the search-space at startup.

ORFEL, as shown in Algorithm 4, leverages parallel vertex-centric graph processing techniques as detailed during Chapter 2. Starting from randomly generated small seed-clusters, ORFEL iteratively grows such clusters by adding both new products, as shown in Algorithm 5, as well as new users, as shown in Algorithm 6, to them when it finds users and products that fit into our lockstep definition and parametrization. Important to consider is that when adding entities to the clusters ORFEL expands upon the delimited δt time-window temporarily allowing users and products that are just outside of the said window to be added to the cluster (technically the temporary window is considered to be double the size of the original window). Immediately after ending each iteration, the algorithm then revisits every cluster, calculating a new center for the time-window based on our objective function, then removing those entries that are not within the desired time-window, as shown in Algorithm 7. Such a mechanism is the main engine behind the algorithm, making it so that we can traverse in the space of potential solutions whilst trying to find the ones that better suit our goal. The algorithm finishes when all clusters are stable, that is, at the end of the iteration, the clusters are the same they were at the beginning. It is also worth noting that some clusters may converge sooner than others, and then the algorithm will skip those already finished clusters in order to maintain the best performance.

Algorithm 4 – ORFEL Algorithm.

```
procedure ORFEL(n, m, ρ, Δt, nSeeds)
    Initialize U[nSeeds], P[nSeeds]                    ▷ Initial Seeding
    repeat
        U' = U
        P' = P
        for each product p in |V| do
            P = updateProducts(p)
        end for
        for each user u in |V| do
            U = updateUsers(u)
        end for
        endIteration()
    until U' = U and P' = P
    return [U, P]
end procedure
```

Dataset	# Users	#Products	# Total nodes	# Edges
Amazon.FineFoods	256,059	74,258	330,317	568,454
Amazon.Movies	889,176	253,059	1,142,235	7,911,684
Synthetic.C	2,000,000	8,000,000	10,000,000	100,000,000

Table 3 – Datasets.

Source: Gimenes, Cordeiro and Rodrigues-Jr (2016).

3.3.2 Results

We ran several experiments to understand how ORFEL performs in both the real world as well as in controlled conditions. Table 3 presents the datasets that were used during our experimentation process, two Amazon datasets, one for the Fine Foods section, and one for the Movies section of the website, reviews are scored between 1 and 5 stars. We also used a synthetic dataset using NetworkX (HAGBERG; SCHULT; SWART, 2008) bipartite graph generator as well as larger versions of the dataset for scalability tests. ORFEL was implemented within the GraphChi (KYROLA; BLELLOCH; GUESTRIN, 2012) framework, all of the necessary components for full reproducibility can be found at <www.icmc.usp.br/pessoas/junio/ORFEL/index.htm>.

The first aspect we analyzed for our algorithm is the ability to detect locksteps of different sizes. To be able to measure it, we generated artificial attacks with varying sizes whilst fixating the algorithm's parameters to see how the algorithm behaves. We appended artificial attacks to the data with sizes varying from 10 users and 5 products to 1,000 users and 500 products. This allowed us to observe the percentage of attacks caught for each configuration of the algorithm – in Figure 3a, ($n = 10$, $m = 5$, $\rho = 0.8$, $nSeeds = 1000$); in Figure 3b, ($n = 50$, $m = 25$, $\rho = 0.8$, $nSeeds = 1000$); and, in Figure 3c, ($n = 50$, $m = 25$, $\rho = 0.8$, $nSeeds = 3000$). These results show that larger attacks are often more easily detected, as well as the fact that if we are trying to

Algorithm 5 – updateProducts

```
procedure UPDATEPRODUCTS(vertex)
    for each Lockstep c ∈ C do
        reviews ← U[c].edges ∩ vertex.edges
        timeCenter ← avgtime(reviews)
        for each edge e in reviews do
            if |e.time − timeCenter| > Δt and λ(e.weight) then
                reviews = reviews − {e}
            end if
        end for
        if |P[c]| < m then
            if (|reviews|/|U[c]|) ≥ ρ then
                P[c] = P[c] ∪ {vertex}
            end if
        else
            for each product p ∈ P[c] do
                if p.reviews ⊂ reviews then
                    swap = p
                end if
            end for
            P[c] = (P[c] − {swap}) ∪ {vertex}
        end if
    end for
end procedure
```

Algorithm 6 – updateUsers

```
procedure UPDATEUSERS(vertex)
    for each Lockstep c ∈ C do
        reviews ← P[c].edges ∩ vertex.edges
        for each edge e in reviews do
            pCenter ← avgtime((u, e.vertex), u ∈ U[c])
            if |e.time − pCenter| > Δt and λ(e.weight) then
                reviews = reviews − {e}
            end if
        end for
        if (|reviews|/|P[c]|) ≥ ρ then
            U[c] = U[c] ∪ {vertex}
        end if
    end for
end procedure
```

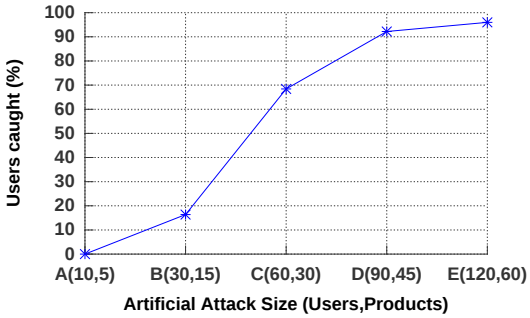
detect attacks that are small or very similar in magnitude to the chosen m and n parameters, the algorithm has more difficulty to find them. This experiment presents a general overview of how the algorithm behaves in relation to different parameters and attack scenarios.

Another interesting question we wanted to answer for ORFEL is how the number of seeds affects the ability of the algorithm to find the attacks. We then proposed the following experiment

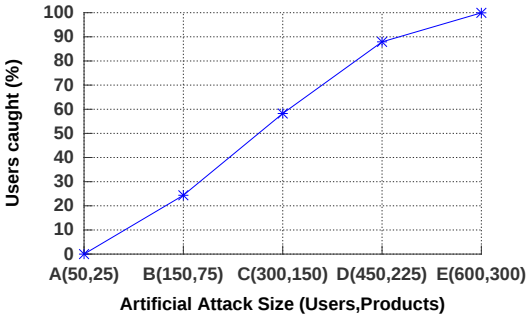
Algorithm 7 – endIteration

```
procedure ENDITERATION
    for each Cluster c ∈ C do
        for each product p ∈ P[c] do
            Sort U[c] by the time of the reviews
            Scan sorted U[c] for the 2Δt-subset that maximizes the number of reviews
            Remove the users from U[c] that are not in the subset
        end for
    end for
end procedure
```

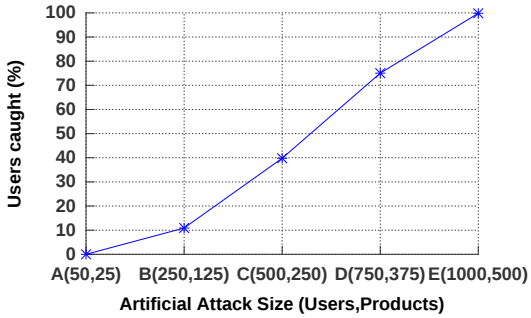
Figure 3 – Experiments of efficacy: the percentage of attacks caught versus the size of the artificially generated attacks. Parameters are described as (n,m,ρ,nSeeds).



(a) Amazon.FineFoods - Parameters (10,5,0.8,1000)



(b) Amazon.Movies - Parameters (50,25,0.8,1000)



(c) Synthetic.C - Parameters (50,25,0.8,3000)

Source: Gimenes, Cordeiro and Rodrigues-Jr (2016).

where we introduce 20 artificial attacks in our datasets and then varied the number of seeds given to the algorithm. Figure 4 shows the efficacy of the algorithm, that is, the percentage of attacks caught versus the number of seeds, for this run parameters [n,m,ρ,AttackSize(Users,Products)] were set as: Synthetic.C [50,25,0.8,(750,375)]; Amazon.Movies [50,25,0.8,(500,250)]; Amazon.FineFoods [10,5,0.8,(50,25)].

With this experiment, we verified that ORFEL identified more than 95% of the attacks in three datasets of different sizes as we started it with enough seeds. This also shows that the algorithm is capable of dealing with different attack sizes given the proper selection of parameters, and is, therefore, able to perform well regardless of specific particularities of different

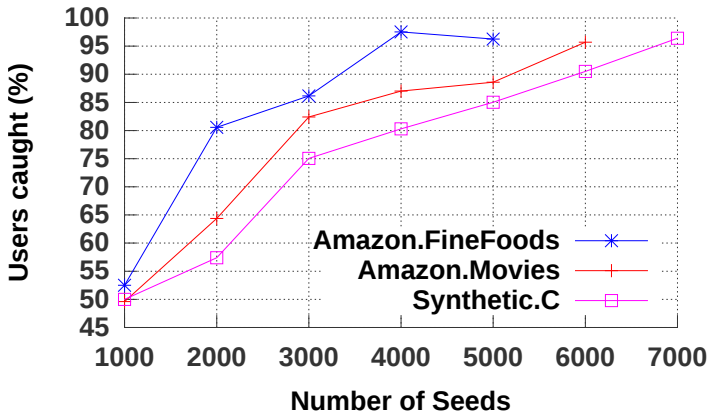


Figure 4 – Efficacy of attacks caught versus number of seeds.

Source: Gimenes, Cordeiro and Rodrigues-Jr (2016).

domains and scenarios. It is also important to note that ORFEL does so while complying to the vertex-centric asynchronous parallel processing paradigm, meaning that it keeps high-scalability, seamless parallelism and high efficiency due to the asynchronicity of the value updates.

Finally, we also ran ORFEL in both Amazon real-world networks without inserting any artificial attacks. We selected that our parameters for a suspicious behavior would be comprised of at least 20 users, recommending 6 products over less than a week. ORFEL was able to detect 37 suspicious clusters which upon further manual analysis revealed themselves to be caused an interesting yet potentially problematic design decision by amazon. All of the locksteps we found were caused by Amazon's policy of merging user reviews for different flavors or sizes of a determined food product and also different versions of the same movie whilst giving them different id's. Such findings, while not exactly considered illegitimate promotion or defamation attacks reveal an important oversight in design that could cause users to be reading a review for a specific product that was actually written by someone who purchased a similar but not quite the same product.

Additionally, as previously stated one of the most important premises for ORFEL is its scalability. Therefore we also tested our algorithm's runtime for different sized datasets as shown in Figure 5. We excluded preprocessing times and took the average time for 3 separate runs. ORFEL manages to process 1 billion edges stored on a mechanical disk within 143 min. ($\approx$2.38 hour) , and 78 min. ($\approx$1.3 hours) using a solid-state disk. We consider this performance to be very efficient given that previous work Beutel *et al.* took $\approx$0.5 hours to do the same processing whilst fully utilizing a computational cluster composed of one thousand machines on top of MapReduce, whereas ORFEL ran in a single commodity machine with an i7-4770 processor and 16GB of RAM, with a 7200RPM HDD and a 450MB/s SSD. Our approach allows for a more efficient framework where multiple nodes are being processed simultaneously, exploiting the inherent parallelism of the algorithm without the need for specific infrastructure while still

allowing for linear scalability.

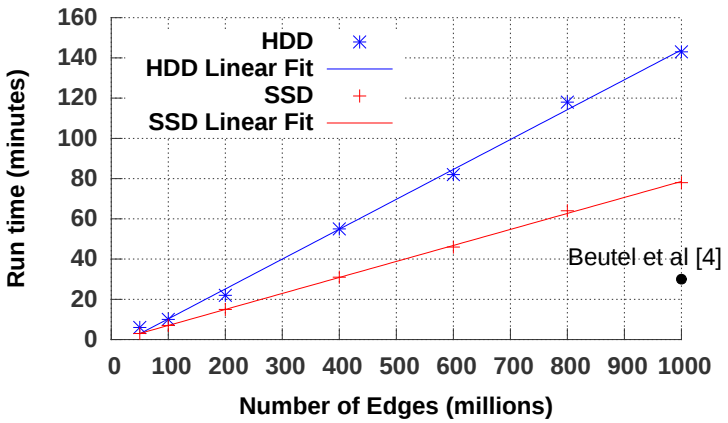


Figure 5 – Experiments of scalability over the number of edges.

Source: Gimenes, Cordeiro and Rodrigues-Jr (2016).

We also present ORFEL's performance when considering increasing amounts of starting seeds. As previous experiments showed that different scenarios may require different amounts of seeds; so we verify that our algorithm is also able to scale linearly against the number of seeds as shown in Figure 6. We ran ORFEL starting with 100 seeds with up to 5000, the program took 10 minutes to run with 100 seeds and 298 min with 5000 seeds. We chose a 100 million edges graph to be used in this experiment.

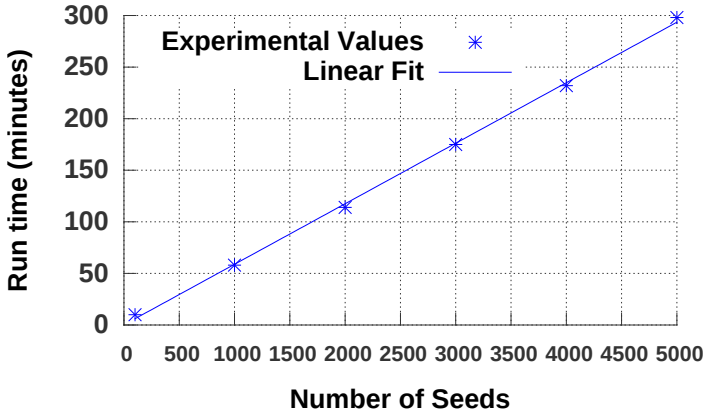


Figure 6 – Experiments of scalability over the number of seeds.

Source: Gimenes, Cordeiro and Rodrigues-Jr (2016).

ORFEL extends the state-of-the-art for the problem of lockstep detection in the following ways:

1. **Novel algorithmic paradigm**: we introduce the first *vertex-centric* algorithm able to spot *lockstep behavior* in Web-scale graphs using asynchronous parallel processing; vertex-

centric processing is a promising paradigm that still lacks algorithms specifically tailored to its *modus operandi*;

2. **Scalability and accuracy**: we tackle the problem for billion-scale graphs in one *single* commodity machine, achieving efficiency that is comparable to that achieved by state-of-the-art works on large *clusters* of computers, whilst obtaining the same efficacy;

3. **Generality of scope:** we tackle the problem for real weighted graphs ranging from social networks to e-commerce recommendation, expanding the state-of-the-art of lockstep semantics to discriminate defamation *and* illegitimate promotion.

With the increasing importance of security and reliability of online services, ORFEL serves a relevant cause, one that can cause high damage both to customers and vendors if malicious and artificial recommendations affect product sales, marketing arrangements and the overall credibility of the ecosystem. Furthermore, we state that while we worked mainly with recommendation systems, our methodology can be promptly extended to other domains. Especially given the high scalability of the algorithm due to the asynchronous vertex-centric parallel processing technique applied. The modular definition of the lockstep behavior, as well as the versatility of the algorithm's parameters, allow for other applications such as social network analysis and scientific citations networks.

3.4 Collaborative Works

3.4.1 Identifying urban inconsistencies

Here we discuss our collaborative work with *Gabriel Spadon* entitled *"Identifying Urban Inconsistencies via Street Networks"* (SPADON; GIMENES; RODRIGUES-JR, 2017). One of the interesting natural occurrences of graph-like data in our everyday lives is the very cities we live in. Possible modeling of our cities considers streets as edges and their intersection as nodes in a network. This allows for a multitude of processes and techniques to be applied to such structures. In our methodology we focused on the topological characterization of these networks, more specifically, we identify *urban inconsistencies*, which are locations of interest in the cities that lack efficient access from or to other regions. By analyzing inconsistencies in a real-world city we discuss urban planning and the impact of improper facility placements.

3.4.1.1 Problem

Preparing the networks

One of the challenges in analyzing street-networks is to obtain a functional representation of the city, that can correctly model the underlying characteristics we are interested in investigating. To do so we used maps provided by the Open Street Map (OpenStreetMap contributors,

2017) project. Although when following the representation previously mentioned of considering streets as edges and their intersections as nodes we found that several real-world structures that are commonly found in cities did not produce an efficient representation in the graph, for instance, roundabouts added a lot of unnecessary clutter to our representation, as well as additional redundancy. Therefore we devised an algorithm that preprocesses the street-networks removing unwanted nodes and edges, whilst improving the representation of the city to be as efficient as possible.

Defining the inconsistencies

The be able to define the inconsistencies we are investigating, we first define a set $\mathbb{F}$ of facilities (nodes) of interest, and then devise two related groups of nodes for each of the facilities: the ones that are closest to the facility when considering the Euclidean distance and another with the nodes that are the closest to the facility via the network-distance metric.

Perimeter set.

The distance d_{ij}^{E} is measured between nodes i and j over the Earth's surface.

$$\mathtt{V}_f^{\mathtt{E}} = \{v \in V \mid d_{vf}^{E} < d_{vf'}^{E} \; \forall \; f \in \mathbb{F},\; f' \in \mathbb{F},\; f \neq f'\} \tag{3.11}$$

It is important to note that given the set of nodes V and a set of facilities $\mathbb{F}$, every node $v \in V$ belongs to the perimeter set of a single facility $f \in \mathbb{F}$.

Network-distance set.

For this set d_{ij}^{N} is the network distance from node i to node j.

$$\mathtt{V}_f^{\mathtt{N}} = \{v \in V \mid d_{vf}^{N} < d_{vf'}^{N} \; \forall \; f \in \mathbb{F},\; f' \in \mathbb{F},\; f \neq f'\} \tag{3.12}$$

Similarly to the first set, every node $v \in V$ belongs to the network set of a single facility $f \in \mathbb{F}$. Given that the network-distance set is directed, it is important to note that the *network-distance set TO a given facility f* might not be the same as the *network distance set FROM a given facility f*.

Finally, with these definitions, we can then consider as a *urban inconsistency* those nodes that are closest to a given facility according to one metric and closed to another facility according to the other measurement. This methodology allows us to consider the inconsistencies as set operations and to devise an algorithm capable of identifying such anomalies.

3.4.1.2 Results

In this section, we present our results and analysis of the *urban inconsistencies* our methodology found in the real-world network derived from the city of São Carlos.

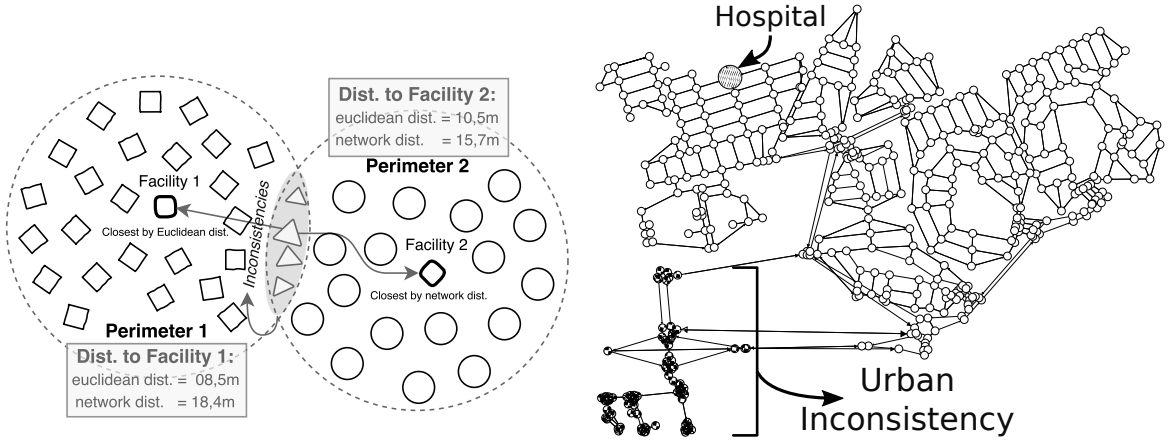
Urban inconsistencies

In order to evaluate our algorithm's capability of finding facilities that may have their location considered a *urban inconsistency* we have analyzed the street-network for the city of São Carlos. We considered 3 types of facilities - police stations, hospitals and schools as shown in Table 4. The first thing we notice is that the number of inconsistencies found is related to the number of facilities analyzed, that is, the more facilities we have of a given category, the more inconsistencies we observe; that happens due to the higher number of overlapping boundaries, where inconsistencies are more likely to occur. Figure 7a presents a depiction of a generic inconsistency example, showcasing an example of inconsistency and its relation to the perimeter boundaries. Whereas Figure 7b presents a real inconsistency found in the city of São Carlos with regards to the location of a Hospital.

Urban Inconsistencies																	
i-th Facility	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	**Total**
Police St.	32	4	86	191	—	—	—	—	—	—	—	—	—	—	—	—	**342**
Hospital	13	2	12	19	30	49	145	39	12	43	72	95	28	—	—	—	**559**
Schools	15	77	43	71	114	3	8	15	78	51	38	15	56	8	60	11	**663**

Table 4 – The inconsistencies from the city of Sao Carlos concerning public facilities.

Figure 7 – The identification of urban inconsistencies.



(a) A sub-graph of the city. (b) Post-processed sub-graph.

Source: Spadon, Gimenes and Rodrigues-Jr (2017).

In this work we focused on identifying low-access regions through a methodology able to analyze urban structures considering a set of facilities. We proposed a set of formalisms for the problem of identifying *urban inconsistencies*, as well as two algorithms: one for the

preprocessing of street-networks to remove unnecessary and redundant information, such as malformed roundabouts, and the other to search and identify the inconsistencies.

Our main contributions include: **(i)** a novel concept based on critical problems in the urban design, which are caused by potentially misplaced facilities in a city; **(ii)** new frameworks to preprocess and prepare maps in the form of street-networks, as well as a method to find and analyze urban inconsistencies; also, **(iii)** the experimental analyses of the discussed methods in the Brazilian city of Sao Carlos.

Whilst it is very challenging to totally eradicate *urban inconsistencies* due to the nature and limitations in the design of modern cities, we believe that the careful analysis and support of specialized tools such as our algorithm can positively impact the overall efficiency of access in a city, opening several new possibilities for the development of more efficient and well-planned cities in the future.

3.4.2 Topological street-network characterization

When we consider street-networks as a mean to represent and analyze a city there are numerous metrics and characteristics that can be extracted from the resulting graphs. We already explored how distances and facility placement can impact a city's mobility and overall efficiency in terms of ease of access and availability of important services (SPADON; GIMENES; RODRIGUES-JR, 2017). However another interesting question that can be asked considering the city models is, can we characterize and then group these cities based on several different measurements? In our second collaborative work with *Gabriel Spadon* (SPADON; GIMENES; RODRIGUES-JR, 2018) we managed to extract characteristics from 645 cities of the Brazilian state of São Paulo, select the most distinctive features and then use those features to cluster cities based on their topological similarity.

3.4.2.1 Problem

In order to properly process and analyze our network data, we followed an Acquisition, Modelling and Computation framework. The first step is to acquire the maps from OpenStreetMaps (OpenStreetMap contributors, 2017) and transform them into street-networks. We then extract several features from the graphs before using a combination of multidimensional projection and cluster analysis to group the cities into partitions.

When deciding which features would be extracted from the networks we considered that global metrics would fit better due to their relationship with the network as a whole, making it more intuitive as means of grouping similar cities. We extracted several metrics and then calculated their pairwise Pearson correlation coefficient (BENESTY *et al.*, 2009) using it to select a subset of metrics that are less correlated, providing a better distinctive power. We began with 29 metrics and chose 9 from that set, these metrics were previously discussed in this chapter

in Section 3.2.1, they are degree distribution entropy, average shortest-path, degree assortativity coefficient, eccentricity, planar network density, central point dominance, two-way streets, and global clustering coefficient.

3.4.2.2 Results

In this section, we present our results and discussions. We divided the experimentation into two parts: first we analyzed how our projections and representations of the cities in São Paulo compare to the population in those cities; second, we turn to the territorial extension of the cities and how that affects how our representations cluster together.

Population size versus topological features

One of the main defining characteristics of a city is the number of inhabitants that live there. In our first experiment we plot the PCA projection of the analyzed cities in a two-dimensional space, despite only extracting topological properties of the cities, we found that those that stand out the most in our projection are the ones that have the higher number of inhabitants, such as Marília, Piracicaba, Campinas and especially São Paulo, it is also worth noting that the city of São Paulo is a clear outlier.

To further analyze the relationship between the topological features and the city's demographics, we decided to remove São Paulo from the dataset in order to better investigate the networks. Expanding on the idea that we can predict the city demographics from its topological characteristics, we measured the correlation between the one-dimensional feature-vector with the actual population for each of the 645 cities. The Pearson correlation for IsoMap was 0.799 and for PCA 0.803, indicating a strong correlation between both variables.

Cluster assignment versus territorial extension

Following our exploratory goals we decided to group our cities using the KMeans algorithm, we tested all numbers of clusters from 2 to 644, aiming to maximize the Silhouette score while keeping the Dunn index above one. We, therefore, found that the best partition for our data under said condition is to have 2 clusters. With a final Silhouette score of 0.59 and Dunn index of 1.10.

Finally, we decided to analyze the reasoning behind the better partition being the 2-clusters setup. We further investigated the demographics of the cities as well as considering the territorial extension this time around. By comparing the territorial extension distribution for the state of São Paulo to the clusters provided by the algorithm we realized that in terms of territorial extension cities in Cluster 1 are: 30.51% tiny-sized, 31.13% small-sized, 25.78% medium-sized, and 12.58% are large-sized; whereas in Cluster 2: 7.59% of are tiny-sized, 6.32% are small-sized, 22.78% are medium-sized, and 63.29% are large-sized. And when it comes to the demographics,

Cluster 1 has 61.20% of the population while Cluster 2 has 38.80%. This allows us to conclude that cities in Cluster 1 tend to be smaller and heavily populated, whereas cities in Cluster 2 are larger and less populated.

Our work was able to confirm that on an overall scope the topological features of a city can shed light into the actual demographics and territorial extension of the city, as well as indications that cities tend to have similar features and group themselves in more ways than simply geographic location or administrative boundaries. Our main contributions in this work can be listed as **(i)** the description of how the network topology is capable of revealing groups of cities with similar characteristics; **(ii)** the correlation analysis between the demography of the cities and their features; and, **(iii)** the discussion of why cities cluster with other cities distant apart instead of with those that they share boundaries with.

3.5 Final considerations

This chapter reviewed an extensive scope of concepts related to graphs/complex networks while presenting practical applications, methodologies, results, and interpretations revolving around those concepts in multiple domains. By presenting both the concepts as well as our works we believe that it consolidates the scope, applicability, and importance of the themes hereby discussed, further contributing and enhancing the body of work comprised in this thesis. The next chapter is going to discuss in detail the Belief Propagation algorithm further stepping towards our proposed solution.

CHAPTER

4

BELIEF PROPAGATION

4.1 Initial Considerations

This chapter presents details of the Belief Propagation algorithm, discussing its history, problems and proposed solutions. The aim of the chapter is to familiarize the reader with the algorithm and its capabilities and limitations.

4.2 Related works

In several circumstances, we are interested in analyzing the propagation of certain information, or belief, in a given community. A belief refers to how individuals influence each other based on their characteristics and preferences. Consider, as an example, the political alignment of individuals in a social network; one might be interested in inferring the political bias of a given node based on its neighbors. The technique known as Belief Propagation (BP) addresses this kind of problem. It is a probabilistic algorithm that estimates the marginal distribution of a given characteristic (belief) for individuals (nodes) that were not observed (measured), conditional to the known information about a set of previously annotated nodes. Such inference is, some times, referred to as *guilty by association*. In the political alignment example, the relationship between the characteristics of the nodes is expected to be homophilic; that is, individuals of a given alignment are expected to be related to individuals with similar biases. Alternatively, it is also possible to model environments where the expected behavior is not homophilic, but heterophilic, in which the relationships are better explained by the *opposites attract* model. It is also possible to have more complex cases, comprising a mix of both paradigms to deal with multiple beliefs at the same time.

To represent such a problem setting, one can benefit from the intuitive modeling offered by graphs, similarly to the modeling seen in domains such as error-correcting codes, statistical physics, computer vision, and artificial intelligence; with applications ranging from fraud

detection to product recommendation (YEDIDIA; FREEMAN; WEISS, 2003).

Beginning with Pearl's original proposal of BP (PEARL, 1982), the algorithm has undergone many improvements to overcome its problems of convergence and scalability. Murphy, Weiss and Jordan (1999) proposed a loopy variation of BP to handle more general scenarios, when graphs are cyclic. In the same line, Yedidia *et al.* (2000) and Yedidia, Freeman and Weiss (2003) proposed a physics-based improvement that allowed the algorithm to converge more than the regular BP, but still without guarantee. With respect to scalability, Felzenszwalb and Huttenlocher (2006) proposed techniques to speed-up the algorithm based on linear messages, and Sudderth *et al.* (2010) presented a nonparametric sampling-based approximation of the method.

Although such proposals augmented the applicability of the algorithm, some issues remained, as pointed out by Elidan, McGraw and Koller (2012), who showed that the loopy BP did not converge as often as expected when used with real-world problems. Additionally, they proposed an asynchronous version of the method tracing relations to its synchronous counterparts.

More recently Mooij and Kappen (2012) started considering the possibility of convergence guarantees on BP by improving on existing bounds and spectral measures. Finally, in 2015, Gatterbauer *et al.* (2015), presented LinBP, a linearized version of BP with exact convergence guarantees, even on cyclic networks, while maintaining similar accuracy. This work is particularly relevant to us since it sets foundations regarding our methodology.

4.3 Algorithm

Belief Propagation (BP) is an algorithm first introduced by Pearl (1982); it computes the approximate marginal probability distribution for an unobserved node by taking into account a set of nodes previously observed (annotated). In our context, we focus on the ability of BP to estimate the degree (probability) according to which a node *believes* in a certain class. It works according to a message-passing computation; in each iteration, each node receives messages from their neighbors, then, based on such information, they update their own beliefs. Iteratively, they propagate messages based on their updated beliefs. This process runs until convergence, that is when the nodes' beliefs cease to alter significantly after any additional iteration.

Along the document, please refer to the List of Symbols for the symbols we use. First, consider a graph $G(V,E)$ and c classes; each node $s \in V$ has two c-dimensional vectors associated to it, $\mathbf{e}_s$ containing the explicit beliefs (priors), and $\mathbf{b}_s$ storing the implicit beliefs (posteriors). Vectors $\mathbf{e}_s$ and $\mathbf{b}_s$ are normalized such that $\sum_{j=0}^{c-1} \mathbf{e}_s(j) = \sum_{j=0}^{c-1} \mathbf{b}_s(j) = 1$. The posterior belief of a node s for class c can be written as:

$$b_s(i) = e_s(i) \prod_{u \in N(s)} m_{us}(i) \tag{4.1}$$

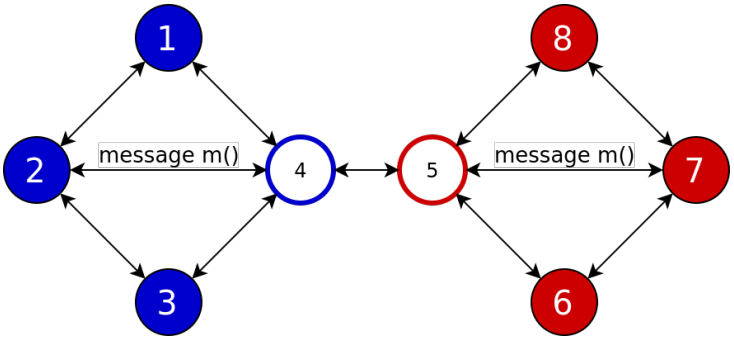
where $m_{us}(i)$ is the message from node u to node s about the i-th class; in the equation, u represents each of the neighbors of node s – denoted by $N(s)$. More generally, for any pair of nodes s and t, the message regarding class i is calculated as:

$$m_{st}(i) = \sum_{j=0}^{c-1} H_{st}(j,i) e_s(j) \prod_{u \in N(s) \setminus t} m_{us}(j) \tag{4.2}$$

where $H_{st}(j,i)$ indicates the relative influence of class j of node s with respect to class i of node t; function H holds for every pair of classes, defining a matrix named *coupling matrix*. The message passing starts with the annotated nodes and proceeds until every node has propagated their beliefs towards their neighbors, comprising an iteration. This version of the algorithm, as stated previously, does not offer any convergence guarantees and has performance problems when considering large graphs.

In order to illustrate the idea behind the algorithm with a simple example, Figure 8 shows a graph with 8 nodes and only 2 belief classes, where we have nodes 1 through 3 with prior explicit *blue* belief and nodes 6 through 8 with prior explicit *red* belief. Since the algorithm works by sending messages iteratively between all connected nodes, in this toy example, it is clear that node 4 would receive more messages indicating that he should *believe* in *blue*, whereas it is the opposite for node 5. It is important to note that 4 and 5 will also exchange messages trying to *convince* each other but it would not be enough to change their classification in this case.

Figure 8 – Toy example for Belief Propagation.



Source: Elaborated by the author.

4.4 Linearized Belief Propagation - *LinBP*

To improve on the common issues of BP, method LinBP was proposed by Gatterbauer *et al.* (2015); it uses the Maclaurin series expansions to replace the product part of the formulation with a summation. According to the LinBP method, due to the linearization of the algorithm, we can write the belief state b for each node, and the message m between nodes, as:

$$\hat{b}_s(i) = \hat{e}_s(i) + \frac{1}{k} \sum_{u \in N(s)} \hat{m}_{us}(i) \tag{4.3}$$

$$\hat{m}_{st}(i) = k \sum_j \hat{H}_{st}(j,i)\hat{b}_s(j) - \sum_j \hat{H}_{st}(j,i)\hat{m}_{ts}(j) \tag{4.4}$$

where $\hat{b}$, $\hat{m}$ and $\hat{H}$ are centered around the parameter $\frac{1}{k}$, meaning that all values are close to $\frac{1}{k}$ and their average is exactly $\frac{1}{k}$. This makes it possible to derive convergence guarantees for the algorithm as well as improves the computational requirements of the algorithm.

This sets the foundation from which we build *VCBP*, by combining the state-of-the-art advances of LinBP with state-of-the-art graph processing techniques to produce a scalable, effective and faster belief propagation algorithm.

4.5 Usage and efficiency

When it comes to using Belief Propagation, it is important to know that the quality of the results depends on the quality of the parameters used to feed the algorithm. In particular, we believe that one of the limitations of the algorithm from a computational standpoint is related to the coupling-matrix - when applying the algorithm to large real-world datasets with several classes it can be very hard to make intuitive sense of the coupling relations between the classes. Particularly when we have complex scenarios comprised of multiple heterophilic and homophilic relations. Additionally, it can be challenging to model the prior labels of known nodes in several real-world domains given that such data is not usually promptly available.

Despite being an algorithm that was proposed several years ago, the algorithm continues to be relevant and new approaches and applications continue to be researched, for instance, Kong, Zhang and Ye (2017) proposes a decentralized BP-based method for multi-agent task allocation, while Mossel *et al.* (2016) works with reconstruction and recovery of block models. But despite its popularity, we find that most works in the literature attack the problem from the statistical point of view and not so many investigate the practical computational characteristics and limitations of the algorithm, especially when it comes to the scalability of the algorithm for very large graphs.

4.6 Final considerations

We presented mathematical formalizations of the Belief Propagation algorithm as well as a retrospect of its evolution along time. The goal of this Chapter is to present the algorithm and lay the foundation of the current state-of-the-art from which the following chapter will pick up to implement our methodology and contributions.

CHAPTER

5

VERTEX-CENTRIC BELIEF PROPAGATION

5.1 Initial considerations

In this chapter, we present our proposed algorithm and processing framework. We further discuss the limitations of the previous solutions and our proposed contributions, including several experiments, datasets, convergence analysis, and a real-world application. This is the culmination of the careful combination of two separate state-of-the-art approaches towards more efficient and scalable computations.

5.2 Problem and Motivation

Although belief propagation addresses many different problems, it is not straightly suited to solve the real-world problems that emerged in the current era, which characterize by strong connectivity. This is due to the fact that the original algorithm (PEARL, 1982) requires the graph to be rooted and acyclic, which are not common characteristics for real-world networks. Improved versions of the algorithm (MURPHY; WEISS; JORDAN, 1999) addressed the problem of dealing with cyclic graphs, but they are not able to cope with graphs as big as those found in social networks, for example. Other works (MURPHY; WEISS; JORDAN, 1999) (YEDIDIA *et al.*, 2000) achieved improvements in the robustness of the belief propagation computation, but they are not able to deal with performance and convergence issues at the same time.

Another aspect is related to the performance of the algorithm when scalability is a need. Most of the works on belief propagation assume that the graph fits in main memory, which may not always be true in the era of Big Data, or in devices with hardware limitations, such as cell phones. With that in mind, options to process these networks often include using supercomputers (CHAU *et al.*, 2011) or relying on distributed processing frameworks such as MapReduce (KANG; CHAU; FALOUTSOS, 2011). Gonzalez, Low and Guestrin (2009) affirms that distributed and synchronous implementations of the algorithm are not efficient

when it comes to the graph-level parallelization that can be achieved using an asynchronous single-machine model. In another work, Gatterbauer *et al.* (2015) proposed *LinBP*, a method that approximates the belief propagation algorithm by using linear transformations while assuring exact guarantees of convergence. Although faster than all the previous BP algorithms, LinBP does not fully explore the performance and scalability potential and limitations of the algorithm in the context of large graphs.

In the context of this work, we provide a more accessible alternative, focusing on using a single consumer-grade computer, which not only allows for a cheaper infrastructure but also manages to provide a simpler framework that can be used by researchers and practitioners. We present the Vertex-Centric Belief Propagation (VCBP), a parallel vertex-centric graph algorithm that uses asynchronous directives to expand on scalability while maintaining convergence guarantees. VCBP is one of the contributions of this thesis, it is designed to run over a single consumer-grade computer to process large graphs. Specifically, the contributions related to CVBP include:

1. **Algorithm:** we designed a vertex-centric algorithm for linearized belief propagation – we used an asynchronous parallel approach to maximize the computation per graph iteration;
2. **Convergence:** being able to do more processing per iteration, we managed to improve the convergence ratio of the algorithm;
3. **Scalability:** we provide an efficient, scalable, and convergence-guaranteed algorithm that can process large-scale graphs in a single commodity machine without loss of precision;
4. **Application:** we discuss new opportunities and difficulties that arise when trying to apply the algorithm to practical problems; particularly, we provide details of VCBP in the task of classification.

5.3 Vertex-centric Belief Propagation - *VCBP*

According to the programming model discussed in Chapter 2, our algorithm has two parts: the main loop as Shown in Algorithm 8, and the *update-function* that is called for each vertex, shown in Algorithm 9. The input to the algorithm consists of the graph G, a list $V_{explicit}$ of vertices with initial known beliefs; a coupling matrix H; and the δ_h scaling factor, which is responsible for assuring the convergence guarantees. Line 2 of Algorithm 8 is where the effective scaling of the coupling matrix happens, before being passed on to the update function. Whereas line 3 is simply the definition of the squared coupling matrix that is going to be used inside the update function and so we do the computation once before entering the main loop so that we do not have to calculate the same values for every vertex. Line 6 represents the call to the update function, the loop continues until we achieve convergence, that is usually a ε indicating the desired maximum relative change in belief values accepted between one iteration and the next.

As seen in Algorithm 9, the actual processing occurs inside the update function. The algorithm starts by initializing the current beliefs for the vertex to 0, lines 5 through 9 comprise the first step of the computation, in which the current vertex consumes incoming messages using their information to update its own beliefs; this is done by considering the influence of the incoming edges using the edges' weights, and the coupling matrix. By combining these values the vertex absorbs all of the information surrounding it. From lines 10 to 15, we apply the *echo cancellation*, a mathematical procedure (GATTERBAUER *et al.*, 2015) that prevents neighboring vertices to constantly induce a given vertex to the same class. After consuming the incoming messages, the vertex then updates its own beliefs and finally propagates these new beliefs to its neighbors as described in lines 16 through 18. Such messages will thereafter be consumed by the neighbors when it is their turn to run the update function, iteratively reaching every node. The output of the algorithm is then a vector of final beliefs for each of the classes for each of the vertices as they were in the moment the algorithm stopped running - typically after convergence.

Algorithm 8 – Vertex-Centric Belief Propagation Algorithm

```
1: procedure VCBP(G.V, V_explicit, H, δ_h)
2:     set H' = δ_h * H
3:     set H'' = H'^2
4:     repeat
5:         for each vertex ∈ G.V do
6:             UPDATE(vertex, H', H'', V_explicit)
7:         end for
8:     until convergence achieved
9: end procedure
```

Convergence criterium and guarantee

Considering that our algorithm is built upon the foundational platform of LinBP (GATTERBAUER *et al.*, 2015), its convergence guarantees remain valid for our algorithm as well. That is, we scale the coupling matrix H via factor eps_h, as shown in Line 2 of Algorithm 8. Following, we include a brief explanation of such convergence guarantees:

$$y^{(l+1)} \leftarrow x + My^{(l)} \tag{5.1}$$

$$VCBP\,converges \Leftarrow ||H|| < \frac{1}{||A||} \tag{5.2}$$

Equation 5.1 shows the Jacobi method iterative solution for linear system $y = (I - M)^{-1}x$; it is known that these updates converge for any initial values for $y_{(0)}$ as long as the spectral radius of M is lower than 1 (GATTERBAUER *et al.*, 2015). Therefore, it is possible to use the fact

Algorithm 9 – Vertex update function of *VCBP*

```
1:  procedure UPDATE(vertex)
2:      Set degree = 0
3:      for each class c ∈ C do                          ▷ initializing vertex values for each class
4:          vertex.value(c) = 0
5:      end for
6:      for each incoming edge e adjacent to vertex do   ▷ processing incoming messages
7:          degree+ = e.weight^2
8:          for each class c_from ∈ C do
9:              for each class c_to ∈ C do
10:                 vertex.value(c_to) += e.weight * e.value(c_from) * H(c_from, c_to)
11:             end for
12:         end for
13:     end for
14:     if vertex ∉ V_explicit then                      ▷ echo cancellation of messages
15:         for each class c_from ∈ C do
16:             for each class c_to ∈ C do
17:                 vertex.value(c_to) −= degree ∗ vertex.value(c_from) ∗ H_2(c_from, c_to)
18:             end for
19:         end for
20:     else                                             ▷ adding explicit value of the vertex
21:         vertex.value(c) += V_Explicit(vertex)(c)
22:     end if
23:     for each outgoing edge e adjacent to vertex do   ▷ sending messages to neighbors
24:         for each class c ∈ C do
25:             e.value(c) = vertex.value(c)
26:         end for
27:     end for
28: end procedure
```

that any norm $||X||$ gives upper bounds to the spectral radius of matrix X to define sufficient convergence conditions, as shown in Equation 5.2. At this point, we are able to choose any norm, such as the Frobenius norm, to calculate sufficient convergence guarantees, which is significantly faster and easier to compute than the spectral radius.

With regard to the convergence criterium, we calculate the ratio between the top-belief (the highest belief for a given node) for subsequent iterations, obtaining the relative change of beliefs between iterations. With this computation, we stop processing once a satisfactory ratio has been achieved. We do it for all the nodes, which must all satisfy the criterium. For our experiments we considered 0.01 as the minimum relative change - less than that between iterations indicates that the convergence was achieved.

Table 5 – Generated Kronecker networks.

Graph	# Nodes	# Edges
1	59,049	1,048,576
2	177,147	4,194,304
3	531,441	16,777,216
4	1,594,323	67,108,864
5	16,777,216	181,776,201
6	33,554,432	363,552,403
7	67,108,864	727,104,806

5.4 Experiments

Here, we compare our algorithm to the state of the art with regard to its efficiency (scalability) and efficacy (accuracy). We are interested in knowing whether our solution improves the scalability of the belief propagation algorithm whilst maintaining competitive accuracy. We used a 3.4 GHz i7 CPU, with 8 cores, 16 GB of main memory, and a 240 GB SSD for storage.

Our implementation of the VCBP algorithm runs over the GraphChi framework (KYROLA; BLELLOCH; GUESTRIN, 2012) – a disk-based vertex-centric graph processing framework, which abstracts the processing of a partitioned graph. All the information required for reproducibility, as well as our source code, can be found at <https://goo.gl/8dQG2a>. Is is important to note that we focused on providing an easy to replicate methodology by making available our code as well as the detailed steps for reproduction.

5.4.1 Datasets

In order to evaluate our algorithm in a controlled environment, while still making sure that the experiments match real-world applications, we use a set of synthetic graphs generated with the Kronecker product method. Proposed by Leskovec *et al.* (2005), graphs generated with the Kronecker product present properties common to real-world datasets. We used the Kronecker generator available in the SNAP library (LESKOVEC; SOSIC, 2014). As the input graph for the generator we used a smaller version of the real-world dataset Polblog (ADAMIC; GLANCE, 2005) that we used in Section 5.5. Table 5 presents the cardinalities of the datasets that we used. For a comprehensive assessing, in Section 5.5, we present experiments on two real-world datasets.

5.4.2 Experimental Setup

In our experiments, we consider a scenario with 3 classes, and the unscaled coupling matrix is shown in Table 6 – unscaled refers to the coupling matrix before we apply the δ_h transformation necessary for the convergence guarantees. We randomly set the initial beliefs for 5% of the nodes; we normalize the beliefs so that they sum up to zero.

Table 6 – Unscaled residual coupling matrix used in our experiments.

	1	2	3
1	0.266667	-0.033333	0.366667
2	0.033333	-0.333333	0.366667
3	-0.233333	0.366667	-0.133333

5.4.3 Accuracy Validation

As mentioned, we are interested in the scalability of our methodology at the same time that we validate its accuracy. We compare the belief assignments computed by our algorithm with those of method LinBP for graphs $1-4$, which were sufficient to demonstrate scalability – the same results extend to graphs $5-7$.

Our results show that VCBP and LinBP agree with respect to the top beliefs assigned for all the test graphs. As desired, both algorithms achieve the same results for nearly 100% of the nodes; with the lowest percentage of accuracy being 99.9% for graph number 4. We attribute this subtle divergence to numerical imprecisions between different implementations, as well as to the random breaking of ties. We consider that such results demonstrate that our algorithm maintains the necessary accuracy standards since it conforms to a baseline implementation widely accepted in the literature. Next, we discuss scalability and convergence improvements.

5.4.4 Scalability

For scalability, we compare our algorithm to the LinBP-SQL implementation. LinBP-SQL is the disk-based implementation of LinBP provided by its very authors. We compare with this version because, just like our work, it is suitable for graphs whose size demands disk storage. We configured PostgreSQL with the default values while following the author's instruction for reproducibility. Additionally we also considered a single-threaded version of our algorithm for comparison to avoid any misconfiguration.

We ran our algorithm for a fixed number of iterations; we ignored the time spent on preprocessing the graphs, such as when loading the datasets into PostgreSQL or sharding the graph in the vertex-centric approach.

We comparatively present the runtimes for algorithms LinBP-SQL and VCBP in Table 7. For graphs 1–4, we compare both methodologies, as shown in Figure 9a; for graphs 1–7, we compare VCBP for one and eight threads in Figure 9b, stressing its asymptotic behavior. As one can see, even when limiting the number of threads to only one, our algorithm is still, at least, 100 times faster. Actually, by adding more processing cores, disk controllers, and execution threads, the performance gains can scale even further. Also, when fully leveraging the parallelism of modern multi-core processors - from one to eight threads, we notice an even greater speed-up, reducing the runtime for the largest analyzed graph almost in half. It is important to emphasize that we are talking about consumer-grade equipment, and also that while adding computational

Table 7 – Runtime (sec) for each graph.

Graph	LinBP-SQL	VCBP-1thread	VCBP-8threads
1	39.04	0.31	0.23
2	179	1.27	0.75
3	826	5.90	3.15
4	5000	34.62	18.69
5	-	152.12	81.27
6	-	340.11	180.35
7	-	759.52	358

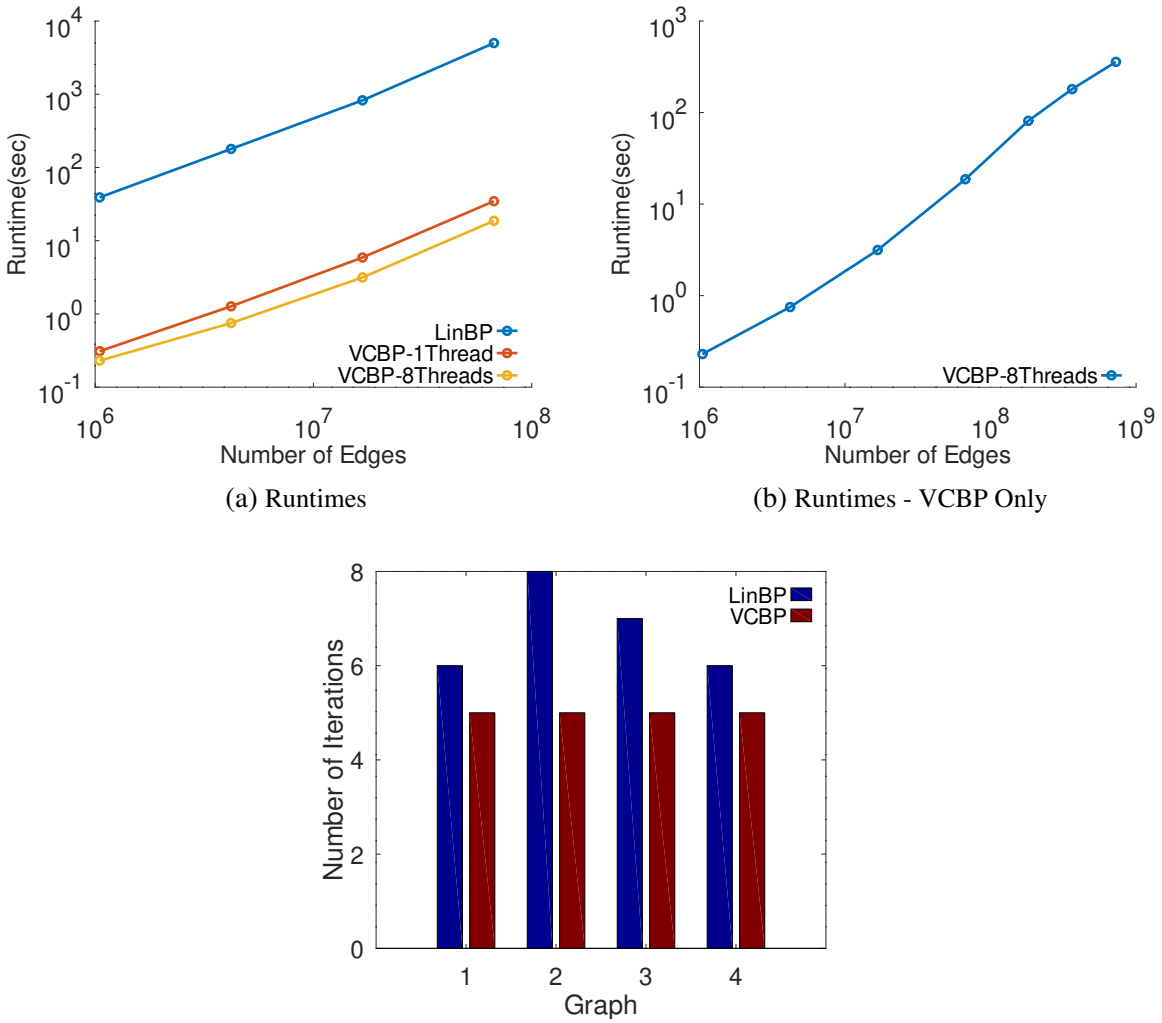
cores can increase the performance further, the disk-access is effectively the bottleneck we are dealing with.

In Figure 9, we analyze the scalability when considering the runtime versus the number of edges in the graph comparatively for VCBP only, with one and eight threads. Our algorithm is able to scale linearly with regard to the number of edges of the network, which allows VCBP to process even very large datasets that may otherwise need supercomputers or computational clusters to be timely processed. We attribute the speed-up to the degree of parallelism brought by an asynchronous vertex-centric approach, while still maintaining the accuracy of a regular method. Figure 9a shows timing experiments for 5 iterations. Figure 9b stresses the asymptotic behavior of our algorithm's scalability by presenting results for graphs orders of magnitude bigger. Figure 9c compares the number of iterations until convergence for each dataset.

5.4.5 Convergence

While performance is a paramount requirement, convergence is also of great concern – the algorithm must converge in a timely manner. In fact, during experimentation, we noticed that the asynchronous processing paradigm also impacted on the convergence behavior of our algorithm. As showed in Figure 9c, VCBP needed fewer iterations to converge than LinBP, despite the same parameters and convergence guarantees. For both algorithms, the convergence criterium considered 0.01 as the minimum relative change to a node's top belief assignment. These findings agree to the work of Elidan, McGraw and Koller (2012), who states that an asynchronous execution of the BP algorithm is guaranteed to be at least as fast as its synchronous counterpart. That is due to the nature of how update functions work, if vertices do not have to wait until the next iteration to receive updated values from their neighbors, this will increase the convergence ratio via effectively compressing multiple synchronous iterations inside a single asynchronous one. Also, since every iteration can be time-consuming, having the algorithm finish a couple of iterations earlier implies in an effectively faster algorithm.

Figure 9 – Experiments comparing LinBP's SQL implementation, with our proposed asynchronous vertex-centric algorithm VCBP.



(a) Runtimes

(b) Runtimes - VCBP Only

(c) Convergence

Source: Elaborated by the author.

5.4.6 Results discussion and future directions

VCBP manages to push the Belief Propagation algorithm performance and efficiency boundaries further, we consider that from our experiments it is possible to conclude that the approach is very effective for this type of algorithm and we attribute this specially to the asynchronicity and overall structure of the processing framework, which aligns naturally with the algorithm's computational requirements.

Another important aspect is that we believe our work can be leveraged in low-memory environments such as embedded and mobile devices, allowing for more efficiency in this scenario where not only the computational power and memory are limited but also other resources such as battery life need to be taken into consideration.

Table 8 – Datasets for the classification experiments.

Graph	# Nodes	# Edges	Classes
PolBlogs (ADAMIC; GLANCE, 2005)	1,490	19,090	Democrat or Republican
Pokec (TAKAC; ZABOVSKY, 2012)	1,632,803	30,622,564	Male or Female

5.5 VCBP in the task of node classification

The use of belief propagation for classification is not a novelty, there are several works in this line, such as CAMLP(YAMAGUCHI; FALOUTSOS; KITAGAWA, 2016) and NetConf(ESWARAN; GÜNNEMANN; FALOUTSOS, 2017). However, while previous works discuss different aspects of the problem, such as the ability to deal with homophily/heterophily, and techniques to expand the versatility of BP, such as including uncertainty in the process; they do not care about the scalability of the algorithms at the same degree of VCBP. In this section, we present early experiments showing how our algorithm can be used in the context of classification. Generally, the lack of convergence guarantees and scalability issues of traditional BP algorithms might limit their use when considering large datasets, therefore we bring attention to this possibility, specifically regarding node classification problems.

Classification based on Belief Propagation

First, we define the problem as:

Definição 2. *Node Classification.* Given a graph $G(V,E)$, with known beliefs w for a subset of nodes $V' \subset V$, and the coupling relation represented by matrix H, find the probability b that a given unannotated node pertains to a class $c \in C$. Finally, classify each node according to its highest class probability.

In order to have Belief Propagation perform classification, it is enough to compute the beliefs in the network until convergence. The procedure stands for a semi-supervised classification since part of the nodes must be previously classified – the final class of each node will refer to the belief with the highest probability.

5.6 VCBP-based classification

To demonstrate the potential of VCBP in the classification task, we ran experiments with the two real-world graphs presented in Table 8.

The first dataset, named PolBlogs, consists of a hyper-link network extracted from political blogs during the 2004 US Election (ADAMIC; GLANCE, 2005). The second one, named Pokec (TAKAC; ZABOVSKY, 2012), is a social network from Slovakia, containing

Table 9 – Accuracy (%) of *VCBP* in the context of classification for two real-world datasets.

Accuracy	PolBlogs	Pokec
VCBP	90.13	70.67
NET-CONF	92.40	75.02

anonymized data from the profiles of its users, such as age, gender, and hobbies. For the experimental setup, we randomly seeded 30% of the nodes with their ground-truth information, and initialized unlabeled nodes to $[\frac{1}{k}, ..., \frac{1}{k}]$, where k is the number of classes. We also used a generic coupling matrix, as defined by Equation 5.3, with $\delta = 0.4$ and $\delta = -0.4$ for homophily and heterophily, respectively.

$$H = \begin{bmatrix} 0.5+\delta & 0.5-\delta \\ 0.5-\delta & 0.5+\delta \end{bmatrix} \tag{5.3}$$

Table 9 shows the accuracy of the algorithm in the task of classification for each of the datasets. These results show that VCBP, the way it is, has the potential for dealing with classification problems; its accuracy is over 90% for PolBlogs and over 70% for Pokec. These numbers are similar to other works specifically designed for classification (ESWARAN; GÜNNEMANN; FALOUTSOS, 2017) (YAMAGUCHI; FALOUTSOS; KITAGAWA, 2016), whilst VCBP maintains its versatility with regard to graphs that may not fit in main memory.

5.7 Future research lines

Given the adequacy of VCBP for the classification task, we provide some lines of work in order to improve its results in terms of accuracy – notice that VCBP already addresses the issue of performance.

While the use of a simple generic coupling matrix can produce competitive results, it is necessary to use more specific and hard-to-obtain prior knowledge about the domain when considering more complex scenarios, with multiple classes and different homophily/heterophily relations between them. Therefore, further research and experimentation are necessary for this front, including, for instance, developing techniques to automatically infer coupling matrices and their classes. Another line of work is to incorporate state-of-the-art advancements into the VCBP's classifier. Advancements such as an uncertainty model, which considers an additional layer of information by including the degree of certainty that a given node has with regard to its own beliefs, allowing for more intuitive results in specific real-world scenarios (ESWARAN; GÜNNEMANN; FALOUTSOS, 2017).

Additionally, it would be interesting to evaluate how VCBP would be able to interact with real-time node classification problems, that is when new nodes are inserted into the dataset. Gatterbauer *et al.* (2015), proposes a variation on the LinBP algorithm, called SBP, that could be

implemented in VCBP's paradigm, allowing for fast and efficient computation of the beliefs of the new nodes. That, coupled with VCBP's faster convergence ratios, could lead to a reliable and efficient way to address such problems.

5.8 Final considerations

In this chapter, we discussed our methodology and proposed solutions to the scalability, convergence and efficiency problems related to the Belief Propagation algorithm. We presented experiments in both synthetic and real-world datasets, discussed problems and difficulties that can arise from a practical standpoint and also presented future research opportunities. The next chapter is going to further discuss our contributions and final thoughts.

CHAPTER

6

CONCLUSIONS

6.1 General considerations

In our project, we managed to investigate several properties and algorithms related to the analysis of complex networks, while achieving significant contributions both in terms of exploratory mining as well as in terms of the scalability and efficiency of the proposed methodologies. Chapter 3 includes three of our contributions: ORFEL is a novel algorithm that not only expanded upon the formalization of the lockstep detection problem but also dealt with scalability problems by leveraging vertex-centric asynchronous parallel processing techniques to provide a cost-effective fraud detection framework for bipartite graph domains. Our results included the analysis of real-world recommender system networks as well as thorough experimentation under controlled synthetic conditions. We also collaboratively explored street-network domains, we proposed preprocessing techniques to obtain satisfactory data from the OpenStreetMap project; delineated the concept of *urban inconsistencies* whilst providing an algorithm to find such ill-devised facility placement potential problems; additionally we investigated the relationship between topological metrics from street-networks and real-world city characteristics such as demographic data and territorial extension, including how those cities group themselves regardless of administrative borders.

Further, in Chapter 5 we showed that it is possible to improve the scalability and usability of the BP algorithm by combining the state-of-the-art BP theory with state-of-the-art graph processing techniques. By taking full advantage of the natural parallel characteristics of the BP algorithm and pairing it with disk-based vertex-centric asynchronous graph processing, we propose a novel algorithm that achieves superior scalability when compared to previous works, even for scenarios where the network does not fit in main memory. Our methodology also culminated in an improved convergence ratio as shown in our analyses, which can be leveraged for time-critical applications pushing the limits of the algorithm with newer technologies while also opening up opportunities for new domains and computational tasks.

VCBP provides a scalable framework for belief propagation tasks in a single computer, building upon the theoretical progress of the linearized Belief Propagation principle. The extent to which such findings can enable previously unexplored tasks is yet to be understood; here, convergence and performance limitations played a significant role in how the problem was exploited. We believe that this makes belief-propagation-based algorithms a solid choice for researchers and practitioners working with network data and willing to explore new possibilities.

We also discussed the task of classification for the Belief Propagation algorithm, presenting experiments and directions for future work so to improve the algorithm and its potential for classification. Finally, our approach brings attention to the yet not-fully explored potential of single-machine parallel processing algorithms. In contrast to works that rely on supercomputers and computational clusters to process large graphs, our methodology offers an alternative that truly benefits from multiple-cores and concurrent programming, reaching results that compare to equipment way more costly and robust. We hope our contributions to foster a more cost-efficient and accessible data-mining scenario.

6.2 Hypothesis

With these contributions, we successfully achieved results in accordance with our general hypothesis, which we reproduce here:

General Hypothesis: *The use of iterative vertex-centric asynchronous parallel processing techniques over multi-core computational architectures can lead to the development of efficient algorithms capable of revealing interesting patterns in large scale networks for real-world domains, such as social networks, recommendation systems, and citation networks.*

The results observed in the paper *Vertex Centric Asynchronous Belief Propagation Algorithm for Large-Scale Graphs* (GIMENES; GUALDRON; RODRIGUES-JR, 2016) demonstrated the successful combination of the Linearized Belief Propagation Principle with the vertex-centric asynchronous parallel processing technique. Along with the results presented in the paper *ORFEL* (GIMENES; CORDEIRO; RODRIGUES-JR, 2016), we achieved highly-scalable graph-processing methods as experimented on datasets from several domains.

6.3 Scientific Production

Published

- Gabriel Gimenes, Robson Cordeiro, Jose Rodrigues-Jr (2016) ORFEL: Efficient detection of defamation or illegitimate promotion in online recommendation Elsevier Information Sciences 379: 10. 274 - 287. (GIMENES; CORDEIRO; RODRIGUES-JR, 2016)

- Gabriel Gimenes, Hugo Gualdron, Jose Rodrigues-Jr (2016) Vertex Centric Asynchronous Belief Propagation Algorithm for Large-Scale Graphs In IEEE 16th International Conference on Data Mining Workshops 93-98 IEEE Press. (GIMENES; GUALDRON; RODRIGUES-JR, 2016)

- Gabriel Spadon, Gabriel Gimenes, Jose Rodrigues (2017) Identifying Urban Inconsistencies via Street Networks In Procedia Computer Science Edited by Elsevier, vol 108. 18–27. (SPADON; GIMENES; RODRIGUES-JR, 2017)

- Gabriel Spadon, Gabriel Gimenes, Jose Rodrigues-Jr (2018) Topological Street-Network Characterization Through Feature-Vector and Cluster Analysis In: Computational Science – ICCS 2018 274-287 Springer International Publishing. (SPADON; GIMENES; RODRIGUES-JR, 2018)

Submitted

- Gabriel Gimenes, Gabriel Spadon, Jose Rodrigues (2019) VCBP - Parallel Asynchronous Vertex-centric Belief Propagation Algorithm (2019) In: ACM Trans. Knowl. Discov. Data

6.4 Mass-media divulgation

- Local Television Report *EPTV* <http://tiny.cc/hwjnfz>

- Local Television Report *Record* <http://tiny.cc/jxjnfz>

- *FAPESP* Report <http://tiny.cc/9xjnfz/>

6.5 Future Work

While we have discussed further directions for our main algorithm in Chapter 5, from a broader perspective, the goals pursued in this thesis are aligned with problems related to computational problems from the perspective of efficiency. That is, we believe that in several fields of knowledge, the society is realizing that while some solutions may work from a practical standpoint, it is also important to consider the implications and long-term sustainability of the implemented systems in face of the ever-increasing volumes of data. Therefore, we reinforce the importance of developing and researching large-scale data mining and graph processing techniques that are available more ubiquitously, without the need for specific, and potentially expensive, infrastructure. Whilst also making sure that we are leveraging our readily available computational power from commodity-machines in the best way possible.

BIBLIOGRAPHY

ADAMIC, L. A.; GLANCE, N. The political blogosphere and the 2004 u.s. election: Divided they blog. In: **Proceedings of the 3rd International Workshop on Link Discovery**. New York, NY, USA: ACM, 2005. (LinkKDD '05), p. 36–43. ISBN 1-59593-215-1. Available: <http://doi.acm.org/10.1145/1134271.1134277>. Citations on pages 71 and 75.

AGGARWAL, C.; SUBBIAN, K. Evolutionary network analysis: A survey. **ACM Computing Surveys (CSUR)**, ACM, v. 47, n. 1, p. 10, 2014. Citation on page 27.

BENESTY, J.; CHEN, J.; HUANG, Y.; COHEN, I. Pearson correlation coefficient. In: **Noise reduction in speech processing**. [S.l.]: Springer, 2009. p. 1–4. Citation on page 57.

BEUTEL, A.; XU, W.; GURUSWAMI, V.; PALOW, C.; FALOUTSOS, C. Copycatch: stopping group attacks by spotting lockstep behavior in social networks. In: **Proceedings of the 22nd international conference on World Wide Web**. [S.l.: s.n.], 2013. p. 119–130. Citation on page 45.

BRANDT, C.; LESKOVEC, J. Status and friendship: mechanisms of social network evolution. In: INTERNATIONAL WORLD WIDE WEB CONFERENCES STEERING COMMITTEE. **Proceedings of the companion publication of the 23rd international conference on World wide web companion**. [S.l.], 2014. p. 229–230. Citation on page 27.

CAPELLI, L. A.; HU, Z.; ZAKIAN, T. A.; BROWN, N.; BULL, J. M. ipregel: Vertex-centric programmability vs memory efficiency and performance, why choose? **Parallel Computing**, Elsevier, v. 86, p. 45–56, 2019. Citation on page 35.

CHAU, D.; NACHENBERG, C.; WILHELM, J.; WRIGHT, A.; FALOUTSOS, C. Polonium: Tera-scale graph mining and inference for malware detection. In: **SIAM International Conference on Data Mining**. [S.l.: s.n.], 2011. v. 2. Citations on pages 30 and 67.

CHENG, J.; ADAMIC, L.; DOW, P. A.; KLEINBERG, J. M.; LESKOVEC, J. Can cascades be predicted? In: INTERNATIONAL WORLD WIDE WEB CONFERENCES STEERING COMMITTEE. **Proceedings of the 23rd international conference on World wide web**. [S.l.], 2014. p. 925–936. Citation on page 27.

CLAUSET, A.; NEWMAN, M. E.; MOORE, C. Finding community structure in very large networks. **Physical review E**, APS, v. 70, n. 6, p. 066111, 2004. Citation on page 43.

CRISTOFARO, E. D.; FRIEDMAN, A.; JOURJON, G.; KAAFAR, M. A.; SHAFIQ, M. Z. Paying for likes? understanding facebook like fraud using honeypots. **arXiv preprint arXiv:1409.2097**, 2014. Citation on page 27.

DEAN, J.; GHEMAWAT, S. Mapreduce: Simplified data processing on large clusters. 2004. Citations on pages 29 and 30.

DHILLON, I. S.; MALLELA, S.; MODHA, D. S. Information-theoretic co-clustering. In: ACM. **Proceedings of the ninth ACM international conference on Knowledge discovery and data mining**. [S.l.], 2003. p. 89–98. Citation on page 45.

DOUCEUR, J. R. The sybil attack. In: **Peer-to-peer Systems**. [S.l.]: Springer, 2002. p. 251–260. Citation on page 45.

ECONOMIST, T. **Data, data everywhere**. 2010. <http://www.economist.com/node/15557443>. Novembro, 2014. Citation on page 25.

ELIDAN, G.; MCGRAW, I.; KOLLER, D. Residual belief propagation: Informed scheduling for asynchronous message passing. **arXiv preprint arXiv:1206.6837**, 2012. Citations on pages 62 and 73.

ESWARAN, D.; GÜNNEMANN, S.; FALOUTSOS, C. The power of certainty: A dirichlet-multinomial model for belief propagation. In: SIAM. **Proceedings of the 2017 SIAM International Conference on Data Mining**. [S.l.], 2017. p. 144–152. Citations on pages 75 and 76.

FELZENSZWALB, P. F.; HUTTENLOCHER, D. P. Efficient belief propagation for early vision. **International journal of computer vision**, Springer, v. 70, n. 1, p. 41–54, 2006. Citation on page 62.

FORTUNATO, S. Community detection in graphs. **Physics Reports**, Elsevier, v. 486, n. 3, p. 75–174, 2010. Citation on page 43.

FREEMAN, L. C. A set of measures of centrality based on betweenness. **Sociometry**, JSTOR, p. 35–41, 1977. Citation on page 41.

____. Centrality in social networks conceptual clarification. **Social networks**, Elsevier, v. 1, n. 3, p. 215–239, 1979. Citation on page 42.

GATTERBAUER, W.; GÜNNEMANN, S.; KOUTRA, D.; FALOUTSOS, C. Linearized and turbo belief propagation. **arXiv preprint arXiv:1406.7288**, 2014. Citation on page 27.

____. Linearized and single-pass belief propagation. **Proceedings of the VLDB Endowment**, VLDB Endowment, v. 8, n. 5, p. 581–592, 2015. Citations on pages 62, 64, 68, 69, and 76.

GIMENES, G.; CORDEIRO, R. L.; RODRIGUES-JR, J. F. Orfel: efficient detection of defamation or illegitimate promotion in online recommendation. **Information Sciences**, p. –, 2016. ISSN 0020-0255. Available: <http://www.sciencedirect.com/science/article/pii/S0020025516307320>. Citations on pages 27, 28, 37, 44, 45, 47, 49, 51, 52, 53, and 80.

GIMENES, G.; GUALDRON, H.; RODRIGUES-JR, J. Vertex centric asynchronous belief propagation algorithm for large-scale graphs. In: **IEEE 16th International Conference on Data Mining Workshops**. IEEE Press, 2016. p. 93–98. Available: <http://conteudo.icmc.usp.br/pessoas/junio/PublishedPapers/Gimenes_et_al_ICDM2016.pdf>. Citations on pages 80 and 81.

GIRVAN, M.; NEWMAN, M. E. Community structure in social and biological networks. **Proceedings of the national academy of sciences**, National Acad Sciences, v. 99, n. 12, p. 7821–7826, 2002. Citation on page 43.

GONZALEZ, J.; LOW, Y.; GUESTRIN, C. Residual splash for optimally parallelizing belief propagation. In: **International Conference on Artificial Intelligence and Statistics**. [S.l.: s.n.], 2009. p. 177–184. Citation on page 67.

GONZALEZ, J. E.; LOW, Y.; GU, H.; BICKSON, D.; GUESTRIN, C. Powergraph: Distributed graph-parallel computation on natural graphs. In: **OSDI**. [S.l.: s.n.], 2012. v. 12, n. 1, p. 2. Citation on page 30.

GUALDRON, H.; CORDEIRO, R.; RODRIGUES-JR, J. F.; HORNG, D.; KAHNG, M.; KANG, U. *et al.* M-flash: Fast billion-scale graph computation using block partition model. **arXiv preprint arXiv:1506.01406**, 2015. Citations on pages 26 and 31.

GÜNNEMANN, S.; GÜNNEMANN, N.; FALOUTSOS, C. Detecting anomalies in dynamic rating data: a robust probabilistic model for rating evolution. In: ACM. **Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining**. [S.l.], 2014. p. 841–850. Citation on page 27.

HAGBERG, A. A.; SCHULT, D. A.; SWART, P. J. Exploring network structure, dynamics, and function using networkx. In: VAROQUAUX, G.; VAUGHT, T.; MILLMAN, J. (Ed.). **Proceedings of the 7th Python in Science Conference**. Pasadena, CA USA: [s.n.], 2008. p. 11 – 15. Citation on page 49.

HAN, W.-S.; LEE, S.; PARK, K.; LEE, J.-H.; KIM, M.-S.; KIM, J.; YU, H. Turbograph: a fast parallel graph engine handling billion-scale graphs in a single pc. In: ACM. **Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining**. [S.l.], 2013. p. 77–85. Citations on pages 26 and 31.

JIANG, M.; CUI, P.; BEUTEL, A.; FALOUTSOS, C.; YANG, S. Catchsync: catching synchronized behavior in large directed graphs. In: ACM. **Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining**. [S.l.], 2014. p. 941–950. Citation on page 27.

KANG, U.; CHAU, D. H.; FALOUTSOS, C. Mining large graphs: Algorithms, inference, and discoveries. In: IEEE. **Data Engineering (ICDE), 2011 IEEE 27th International Conference on**. [S.l.], 2011. p. 243–254. Citations on pages 30 and 67.

KANG, U.; MEEDER, B.; PAPALEXAKIS, E. E.; FALOUTSOS, C. Heigen: Spectral analysis for billion-scale graphs. **Knowledge and Data Engineering, IEEE Transactions on**, IEEE, v. 26, n. 2, p. 350–362, 2014. Citation on page 26.

KANG, U.; TONG, H.; SUN, J.; LIN, C.-Y.; FALOUTSOS, C. Gbase: a scalable and general graph management system. In: ACM. **Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining**. [S.l.], 2011. p. 1091–1099. Citation on page 26.

KANG, U.; TSOURAKAKIS, C.; APPEL, A. P.; FALOUTSOS, C.; LESKOVEC, J. **HADI: Fast diameter estimation and mining in massive graphs with Hadoop**. [S.l.]: Carnegie Mellon University, School of Computer Science, Machine Learning Department, 2008. Citation on page 26.

KANG, U.; TSOURAKAKIS, C. E.; FALOUTSOS, C. Pegasus: A peta-scale graph mining system implementation and observations. In: IEEE. **Data Mining, 2009. ICDM'09. Ninth IEEE International Conference on**. [S.l.], 2009. p. 229–238. Citations on pages 26 and 30.

KONG, Y.; ZHANG, M.; YE, D. A belief propagation-based method for task allocation in open and dynamic cloud environments. **Knowledge-Based Systems**, Elsevier, v. 115, p. 123–132, 2017. Citation on page 64.

KRISHNAMURTHY, B.; GILL, P.; ARLITT, M. A few chirps about twitter. In: **Proceedings of the First Workshop on Online Social Networks**. New York, NY, USA: ACM, 2008. (WOSN

'08), p. 19–24. ISBN 978-1-60558-182-8. Available: <http://doi.acm.org/10.1145/1397735.1397741>. Citation on page 26.

KYROLA, A. Drunkardmob: billions of random walks on just a pc. In: ACM. **Proceedings of the 7th ACM conference on Recommender systems**. [S.l.], 2013. p. 257–264. Citation on page 26.

KYROLA, A.; BLELLOCH, G. E.; GUESTRIN, C. Graphchi: Large-scale graph computation on just a pc. In: **OSDI**. [S.l.: s.n.], 2012. v. 12, p. 31–46. Citations on pages 26, 31, 34, 35, 49, and 71.

LESKOVEC, J.; CHAKRABARTI, D.; KLEINBERG, J.; FALOUTSOS, C. Realistic, mathematically tractable graph generation and evolution, using kronecker multiplication. In: **Knowledge Discovery in Databases: PKDD 2005**. [S.l.]: Springer, 2005. p. 133–145. Citation on page 71.

LESKOVEC, J.; SOSIC, R. **SNAP: A general purpose network analysis and graph mining library in C++**. 2014. Http://snap.stanford.edu/snap. Citation on page 71.

LIN, Y.; RAZA, A. A.; LEE, J.-Y.; KOUTRA, D.; ROSENFELD, R.; FALOUTSOS, C. Influence propagation: Patterns, model and a case study. In: **Advances in Knowledge Discovery and Data Mining**. [S.l.]: Springer, 2014. p. 386–397. Citation on page 27.

LIU, C.; GUO, F.; FALOUTSOS, C. Bbm: bayesian browsing model from petabyte-scale data. In: ACM. **Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining**. [S.l.], 2009. p. 537–546. Citation on page 26.

LLOYD, S. Least squares quantization in pcm. **IEEE transactions on information theory**, IEEE, v. 28, n. 2, p. 129–137, 1982. Citation on page 45.

LOW, Y.; BICKSON, D.; GONZALEZ, J.; GUESTRIN, C.; KYROLA, A.; HELLERSTEIN, J. M. Distributed graphlab: a framework for machine learning and data mining in the cloud. **Proceedings of the VLDB Endowment**, VLDB Endowment, v. 5, n. 8, p. 716–727, 2012. Citation on page 30.

LOW, Y.; GONZALEZ, J.; KYROLA, A.; BICKSON, D.; GUESTRIN, C.; HELLERSTEIN, J. M. Graphlab: A new framework for parallel machine learning. **arXiv preprint arXiv:1006.4990**, 2010. Citation on page 26.

MALEWICZ, G.; AUSTERN, M. H.; BIK, A. J.; DEHNERT, J. C.; HORN, I.; LEISER, N.; CZAJKOWSKI, G. Pregel: a system for large-scale graph processing. In: ACM. **Proceedings of the 2010 ACM SIGMOD International Conference on Management of data**. [S.l.], 2010. p. 135–146. Citations on pages 26 and 30.

MAO, H.-H.; WU, C.-J.; PAPALEXAKIS, E. E.; FALOUTSOS, C.; LEE, K.-C.; KAO, T.-C. Malspot: Multi2 malicious network behavior patterns analysis. In: **Advances in Knowledge Discovery and Data Mining**. [S.l.]: Springer, 2014. p. 1–14. Citation on page 27.

MCAULEY, J. J.; LESKOVEC, J. From amateurs to connoisseurs: modeling the evolution of user expertise through online reviews. In: INTERNATIONAL WORLD WIDE WEB CONFERENCES STEERING COMMITTEE. **Proceedings of the 22nd international conference on World Wide Web**. [S.l.], 2013. p. 897–908. Citation on page 27.

MCGLOHON, M.; BAY, S.; ANDERLE, M. G.; STEIER, D. M.; FALOUTSOS, C. Snare: a link analytic system for graph labeling and risk detection. In: ACM. **Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining**. [S.l.], 2009. p. 1265–1274. Citation on page 30.

MOOIJ, J.; KAPPEN, H. Sufficient conditions for convergence of loopy belief propagation. **arXiv preprint arXiv:1207.1405**, 2012. Citation on page 62.

MOSSEL, E.; NEEMAN, J.; SLY, A. *et al.* Belief propagation, robust reconstruction and optimal recovery of block models. **The Annals of Applied Probability**, Institute of Mathematical Statistics, v. 26, n. 4, p. 2211–2256, 2016. Citation on page 64.

MURPHY, K. P.; WEISS, Y.; JORDAN, M. I. Loopy belief propagation for approximate inference: An empirical study. In: MORGAN KAUFMANN PUBLISHERS INC. **Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence**. [S.l.], 1999. p. 467–475. Citations on pages 62 and 67.

NEWMAN, M. **Networks: an introduction**. [S.l.]: Oxford University Press, 2010. Citation on page 41.

NEWMAN, M. E. Mixing patterns in networks. **Physical Review E**, APS, v. 67, n. 2, p. 026126, 2003. Citation on page 42.

____. Fast algorithm for detecting community structure in networks. **Physical review E**, APS, v. 69, n. 6, p. 066133, 2004. Citation on page 43.

____. Finding community structure in networks using the eigenvectors of matrices. **Physical review E**, APS, v. 74, n. 3, p. 036104, 2006. Citation on page 44.

OpenStreetMap contributors. **Planet dump retrieved from https://planet.osm.org** . 2017. <https://www.openstreetmap.org>. Citations on pages 55 and 57.

PAGE, L.; BRIN, S.; MOTWANI, R.; WINOGRAD, T. The pagerank citation ranking: Bringing order to the web. Stanford InfoLab, 1999. Citations on pages 26, 32, and 42.

PAKHIRA, M. K.; BANDYOPADHYAY, S.; MAULIK, U. Validity index for crisp and fuzzy clusters. **Pattern recognition**, Elsevier, v. 37, n. 3, p. 487–501, 2004. Citation on page 45.

PANDIT, S.; CHAU, D. H.; WANG, S.; FALOUTSOS, C. Netprobe: a fast and scalable system for fraud detection in online auction networks. In: ACM. **Proceedings of the 16th international conference on World Wide Web**. [S.l.], 2007. p. 201–210. Citation on page 30.

PAPALEXAKIS, E.; KANG, U.; FALOUTSOS, C.; SIDIROPOULOS, N.; HARPALE, A. Large scale tensor decompositions: Algorithmic developments and applications. **context**, v. 1, n. b1, p. c1, 2013. Citation on page 27.

PAPALEXAKIS, E. E.; BEUTEL, A.; STEENKISTE, P. Network anomaly detection using co-clustering. In: IEEE COMPUTER SOCIETY. **Proceedings of the 2012 International Conference on Advances in Social Networks Analysis and Mining**. [S.l.], 2012. p. 403–410. Citation on page 45.

PAPALEXAKIS, E. E.; SIDIROPOULOS, N. D. Co-clustering as multilinear decomposition with sparse latent factors. In: IEEE. **Acoustics, Speech and Signal Processing, 2011 IEEE International Conference on**. [S.l.], 2011. p. 2064–2067. Citation on page 45.

PARK, H.-M.; SILVESTRI, F.; KANG, U.; PAGH, R. Mapreduce triangle enumeration with guarantees. In: ACM. **Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management**. [S.l.], 2014. p. 1739–1748. Citation on page 27.

PEARL, J. Reverend bayes on inference engines: A distributed hierarchical approach. In: **AAAI**. [S.l.: s.n.], 1982. p. 133–136. Citations on pages 62 and 67.

PONS, P.; LATAPY, M. Computing communities in large networks using random walks. **J. Graph Algorithms Appl.**, v. 10, n. 2, p. 191–218, 2006. Citation on page 44.

RINGNÉR, M. What is principal component analysis? **Nature biotechnology**, Nature Publishing Group, v. 26, n. 3, p. 303, 2008. Citation on page 44.

RODRIGUEZ, M. G.; LESKOVEC, J.; BALDUZZI, D.; SCHÖLKOPF, B. Uncovering the structure and temporal dynamics of information propagation. **Network Science**, Cambridge Univ Press, v. 2, n. 01, p. 26–65, 2014. Citation on page 27.

ROY, A.; MIHAILOVIC, I.; ZWAENEPOEL, W. X-stream: edge-centric graph processing using streaming partitions. In: ACM. **Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles**. [S.l.], 2013. p. 472–488. Citation on page 26.

____. X-stream: Edge-centric graph processing using streaming partitions. In: **Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles**. New York, NY, USA: ACM, 2013. (SOSP '13), p. 472–488. ISBN 978-1-4503-2388-8. Citations on pages 31, 34, and 35.

SABRIN, K. M.; LIN, Z.; CHAU, D. H. P.; LEE, H.; KANG, U. Mmap: Mining billion-scale graphs on a pc with fast, minimalist approach via memory mapping. Georgia Institute of Technology, 2013. Citation on page 26.

SHAH, N.; BEUTEL, A.; GALLAGHER, B.; FALOUTSOS, C. Spotting suspicious link behavior with fbox: An adversarial perspective. **arXiv preprint arXiv:1410.3915**, 2014. Citation on page 27.

SHIN, K.; KANG, U. Distributed methods for high-dimensional and large-scale tensor factorization. **arXiv preprint arXiv:1410.5209**, 2014. Citation on page 27.

SHNEIDERMAN, B. Building trusted social media communities: A research roadmap for promoting credible content. In: **Roles, Trust, and Reputation in Social Media Knowledge Markets**. [S.l.]: Springer, 2015. p. 35–43. Citation on page 27.

SHVACHKO, K.; KUANG, H.; RADIA, S.; CHANSLER, R. The hadoop distributed file system. In: IEEE. **Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on**. [S.l.], 2010. p. 1–10. Citation on page 26.

SIDIROPOULOS, N.; PAPALEXAKIS, E.; FALOUTSOS, C. Parallel randomly compressed cubes: A scalable distributed architecture for big tensor decomposition. **Signal Processing Magazine, IEEE**, IEEE, v. 31, n. 5, p. 57–70, 2014. Citation on page 27.

SPADON, G.; GIMENES, G.; RODRIGUES-JR, J. Identifying urban inconsistencies via street networks. In: ELSEVIER vol . (Ed.). **Procedia Computer Science**. [S.l.: s.n.], 2017. p. 18;27. Citations on pages 37, 54, 56, 57, and 81.

____. Topological street-network characterization through feature-vector and cluster analysis. In: **Computational Science; ICCS 2018**. [S.l.]: Springer International Publishing, 2018. p. 274–287. Citations on pages 37, 44, 45, 57, and 81.

SUDDERTH, E. B.; IHLER, A. T.; ISARD, M.; FREEMAN, W. T.; WILLSKY, A. S. Nonparametric belief propagation. **Communications of the ACM**, ACM, v. 53, n. 10, p. 95–103, 2010. Citation on page 62.

TAKAC, L.; ZABOVSKY, M. Data analysis in public social networks. In: **International Scientific Conference and International Workshop Present Day Trends of Innovations**. [S.l.: s.n.], 2012. v. 1, n. 6. Citation on page 75.

TAMERSOY, A.; ROUNDY, K.; CHAU, D. H. Guilt by association: large scale malware detection by mining file-relation graphs. In: ACM. **Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining**. [S.l.], 2014. p. 1524–1533. Citation on page 30.

TENENBAUM, J. B.; SILVA, V. D.; LANGFORD, J. C. A global geometric framework for nonlinear dimensionality reduction. **science**, American Association for the Advancement of Science, v. 290, n. 5500, p. 2319–2323, 2000. Citation on page 44.

WEST, R.; PASKOV, H. S.; LESKOVEC, J.; POTTS, C. Exploiting social network structure for person-to-person sentiment analysis. **arXiv preprint arXiv:1409.2450**, 2014. Citation on page 27.

XIE, Z.; ZHU, S. Grouptie: toward hidden collusion group discovery in app stores. In: ACM. **Proceedings of the 2014 ACM conference on Security and privacy in wireless & mobile networks**. [S.l.], 2014. p. 153–164. Citation on page 27.

YAMAGUCHI, Y.; FALOUTSOS, C.; KITAGAWA, H. Camlp: Confidence-aware modulated label propagation. In: SIAM. **Proceedings of the 2016 SIAM International Conference on Data Mining**. [S.l.], 2016. p. 513–521. Citations on pages 75 and 76.

YEDIDIA, J. S.; FREEMAN, W. T.; WEISS, Y. Understanding belief propagation and its generalizations. **Exploring artificial intelligence in the new millennium**, v. 8, p. 236–239, 2003. Citation on page 62.

YEDIDIA, J. S.; FREEMAN, W. T.; WEISS, Y. *et al.* Generalized belief propagation. In: **NIPS**. [S.l.: s.n.], 2000. v. 13, p. 689–695. Citations on pages 62 and 67.

ZHU, X.; GHAHRAMANI, Z. **Learning from labeled and unlabeled data with label propagation**. [S.l.], 2002. Citation on page 26.


ICMC
USP
SÃO CARLOS