# Block-based and structure-based techniques for large-scale graph processing and visualization

***Hugo Armando Gualdron Colmenares***

**Hugo Armando Gualdron Colmenares**

# Block-based and structure-based techniques for large-scale graph processing and visualization

Master disssertation submitted to the Instituto de Ciências Matemáticas e de Computação – ICMC-USP, in partial fulfillment of the requirements for the degree of the Master Program in Computer Science and Computational Mathematics. *FINAL VERSION*

Concentration Area: Computer Science and Computational Mathematics

Advisor: Prof. Dr. José Fernando Rodrigues Júnior

**USP – São Carlos**
**January 2016**

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

CGualdb

Colmenares, Hugo Armando Gualdron
Block-based and structure-based techniques for large-scale graph processing and visualization / Hugo Armando Gualdron Colmenares; orientador José Fernando Rodrigues Júnior. - São Carlos - SP, 2016.
80 p.

Dissertação (Mestrado - Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional) - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2016.

1. StructMatrix. 2. M-Flash. 3. large-scale graph visualization. 4. billion-scale graph processing. 5. bimodal block partition strategy (BBP). I. Júnior, José Fernando Rodrigues, orient. II. Título.

**Hugo Armando Gualdron Colmenares**

# Tecnicas baseadas em bloco e em estrutura para o processamento e visualização de grafos em larga escala

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências – Ciências de Computação e Matemática Computacional. V*ERSÃO REVISADA*

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientador: Prof. Dr. José Fernando Rodrigues Júnior

**USP – São Carlos**
**Janeiro de 2016**

*For my wife Diana and my child Santiago,*
*Perseverance is stronger than fear.*
*We are going far together.*

# ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my advisor Dr. José Fernado Rodrigues Júnior for his patience, motivation, and immense knowledge. His guidance helped me in all the time of research. My sincere thanks also goes to professor Dr. Duen Horng Chau for his research support during my internship at Georgia Tech University. I would also like to thank FAPESP for financially supporting my research. I thank my fellow labmates in for the inspiring discussions.

Last but not the least, I would like to thank to my wife Diana and my child Santiago for their patience and company, and to my family: my parents, my brother, my sister, my aunts, my uncles, and my cousins, for their time and support to overcome the great challenges of my life.

# RESUMO

GUALDRON, H.. **Block-based and structure-based techniques for large-scale graph processing and visualization**. 2016. 80 f. Master dissertation (Master student Program in Computer Science and Computational Mathematics) – Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos – SP.

Técnicas de análise de dados podem ser úteis em processos de tomada de decisão, quando padrões de interesse indicam tendências em domínios específicos. Tais tendências podem auxiliar a avaliação, a definição de alternativas ou a predição de eventos. Atualmente, os conjuntos de dados têm aumentado em tamanho e complexidade, impondo desafios para recursos modernos de hardware. No caso de grandes conjuntos de dados que podem ser representados como grafos, aspectos de visualização e processamento escalável têm despertado interesse. Arcabouços distribuídos são comumente usados para lidar com esses dados, mas a implantação e o gerenciamento de clusters computacionais podem ser complexos, exigindo recursos técnicos e financeiros que podem ser proibitivos em vários cenários. Portanto é desejável conceber técnicas eficazes para o processamento e visualização de grafos em larga escala que otimizam recursos de hardware em um único nó computacional. Desse modo, este trabalho apresenta uma técnica de visualização chamada *StructMatrix* para identificar relacionamentos estruturais em grafos reais. Adicionalmente, foi proposta uma estratégia de processamento bimodal em blocos, denominada *Bimodal Block Processing* (BBP), que minimiza o custo de I/O para melhorar o desempenho do processamento. Essa estratégia foi incorporada a um arcabouço de processamento de grafos denominado *M-Flash* e desenvolvido durante a realização deste trabalho.Foram conduzidos experimentos a fim de avaliar as técnicas propostas. Os resultados mostraram que a técnica de visualização StructMatrix permitiu uma exploração eficiente e interativa de grandes grafos. Além disso, a avaliação do arcabouço M-Flash apresentou ganhos significativos sobre todas as abordagens baseadas em memória secundária do estado da arte. Ambas as contribuições foram validadas em eventos de revisão por pares, demonstrando o potencial analítico deste trabalho em domínios associados a grafos em larga escala.

**Palavras-chave:** StructMatrix, M-Flash, visualizaço de grafos em larga escala, processamento de grafos em larga escala, processamento bimodal em blocos (BBP).

# ABSTRACT

GUALDRON, H.. **Block-based and structure-based techniques for large-scale graph processing and visualization**. 2016. 80 f. Master dissertation (Master student Program in Computer Science and Computational Mathematics) – Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos – SP.

Data analysis techniques can be useful in decision-making processes, when patterns of interest can indicate trends in specific domains. Such trends might support evaluation, definition of alternatives, or prediction of events. Currently, datasets have increased in size and complexity, posing challenges to modern hardware resources. In the case of large datasets that can be represented as graphs, issues of visualization and scalable processing are of current concern. Distributed frameworks are commonly used to deal with this data, but the deployment and the management of computational clusters can be complex, demanding technical and financial resources that can be prohibitive in several scenarios. Therefore, it is desirable to design efficient techniques for processing and visualization of large scale graphs that optimize hardware resources in a single computational node. In this course of action, we developed a visualization technique named *StructMatrix* to find interesting insights on real-life graphs. In addition, we proposed a graph processing framework *M-Flash* that used a novel, bimodal block processing strategy (*BBP*) to boost computation speed by minimizing I/O cost. Our results show that our visualization technique allows an efficient and interactive exploration of big graphs and our framework M-Flash significantly outperformed all state-of-the-art approaches based on secondary memory. Our contributions have been validated in peer-review events demonstrating the potential of our finding in fostering the analytical possibilities related to large-graph data domains.

**Key-words:** StructMatrix, M-Flash, large-scale graph visualization, billion-scale graph processing, bimodal block partition strategy (BBP).

# LIST OF FIGURES

# LIST OF ALGORITHMS

# LIST OF TABLES

# CONTENTS

CHAPTER

# 1

# INTRODUCTION

## 1.1 Motivation

The computational technology in the XXI century has provided powerful resources to generate digital content, massive access to information, and connectivity between systems, people and electronic devices. This technology produces data in unprecedented scale, having received different denominations: big data, web scale, massive data, planetary or large scale [1]. It is not easy to measure the size of the digital information currently being produced, but some works suggest its size and growth. The Twitter company, for instance, publicly demonstrated the evolution of its social network in 2011 [3]; according to which the time between the first and the billion-th message was of only three years. Recently, a work of Gudinava *et al.* [4] estimates that just a small share of 1 petabyte of the data currently produced is public; and that only 22 percent of this data can be considered useful. This data contains intrinsic and valuable information, and companies like *General Electric* and *Accenture* have shown that data analysis techniques are a differential factor in industrial competitiveness [5].

## 1.2 Problem

A relevant set of the data produced in planetary scale describes relations that can be represented using graphs. Graph representations are useful to analyze and optimize problems in domains such as public politics, commercial decisions, security and social networks. Usual graph processing and visualization systems assume that the graph data fit entirely in the main memory; however, the current size of big graphs can be equivalent to a whole disk or even an array of disks with storage needs up to 100x larger than RAM. For instance, the Twitter graph [6] can be measured in terabyte units; the Yahoo Web graph [7] has more than 1 billion nodes and almost 7 billion edges; and the clickstreams graph [8] reaches petabyte magnitude. Such graphs are a challenge

[1] In this work, we use these terms interchangeably, we emphasis on planetary and large scale.

for data analysis because they require powerful resources for processing, and complex models to achieve tasks with limited resources. This issue happens because graph algorithms usually are based on depth or on breadth processing techniques that are not parallelizable. Existing frameworks, based on vertex-centric [9] [10] and edge-centric [11] processing, tackle with those limitations; but they do not optimize resource utilization, failing in sub-optimal approaches for processing planetary-scale graphs.

## 1.3 Hypotheses

Considering optimization of resources as a cornerstone for processing planetary-scale graphs, and for interactive visualization as a desirable tool in decision-making, we explored two possibilities:

1. we researched on how to detect macro features of very large graphs, mining recurrent patterns like cliques, bi-partite cores, stars and chains; we also analyzed relevant graph domains, characterizing them according to the cardinality, distribution, and relationship of their patterns, leading to the following hypothesis: **relations between recurrent and simple patterns characterize graph domains providing interesting insights through exploratory visualization**;
2. we designed an algorithmic framework that minimizes disk and memory access during vertex-centric and edge-centric processing using only one computational node. We focused on how to minimize I/O communication, taking advantage of hardware capabilities and advanced algorithm design. Accordingly, we worked on validating the following hypothesis: **a framework focused on minimizing I/O communication is able to boost the processing speed of planetary-scale graphs that do not fit in RAM.**

## 1.4 Rationale

Large graph visualization is a research area widely studied [12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23]; however, there are not optimal solutions considering scalability and interaction for many visual tasks [24]. In this research area, visual analysis has two predominant paradigms: *top-down* (global views) and *bottom-up* (local-views). The *top-down* approaches are based on the Shneiderman's mantra overview, zoom and filter, details-on-demand" [25] and they have shown good applicability in many domains [26, 27, 28, 29]; notwithstanding, the creation process of global views is computationally expensive for graphs with millions or billions of vertices and edges. Thus, global views can be cluttered for screens with limited resolution [30] in such a way that simple tasks, like edge navigation or counting, become difficult for users [31]. On the other hand, *bottom-up* approaches [32, 33, 34, 35, 36] have better scalability to show specific regions of the graph, but the selection of initial vertices is a complex task [37]. Addressing the gaps found in these works, one of the lines of this work was to search for approaches considering scalability

(both visual and computational) and interactivity in order to make sense of planetary-scale graphs, as delineated in our first hypothesis. We detail this work in Chapter 2 along with further literature review.

Graph visualization techniques usually require efficient algorithms for graph partitioning, summarization, filtering, clustering or some other task related to graph mining. The computation of such patterns properties, considering planetary-scale graphs, demands distributed and parallel frameworks. The work of Pregel [9], for example, defines a seminal framework that provides a simple programming interface based on message passing; it uses the paradigm "think like vertex" hiding the complexity of distributed systems. Another framework, named GBase [38], uses a MapReduce [39]/Hadoop[40] implementation for graph processing through a block compress partition scheme; in turn, GraphLab [10] introduces an asynchronous, dynamic and graph-parallel scheme for machine learning that permits implementations of many algorithms for graphs; and PowerGraph [41] improves the approach of GraphLab by exploiting power-law distributions – common in graphs of many domains. There are other distributed implementations as the Apache Giraph [42], Trinity [43] and GraphX [44]; but, in general, distributed approaches may not always be the best option, because they can be expensive to build [45] and hard to maintain and optimize.

In addition to the distributed frameworks, single-node processing solutions have recently reached a comparative performance to distributed systems for similar algorithms [46]. These frameworks use secondary memory as an extension of main memory (RAM). GraphChi [45] was one of the first single-node approaches to avoid random disk/edge accesses; it uses a vertex-centric model improving the performance for mechanical disks. Another solution, X-Stream [11] introduced an edge-centric model that achieved better performance over the vertex-centric approaches, it did so by favoring sequential disk access over unordered data. In another way, the FlashGraph framework [47] provides an efficient implementation that considers enough RAM memory for storing vertex values, but its scalability is limited. Among the existing works designed for single-node processing, some of them are restricted to SSDs. These works rely on the remarkable low-latency and improved I/O of SSDs compared to magnetic disks. This is the case of TurboGraph [48] and RASP [49], which rely on random accesses to the edges — not well supported over magnetic disks. In the current research, we have extensively studied these framework in order to design a new methodology that surpasses the performance of all of the previous single-node processing frameworks. In Chapter 3, we provide further details and results that demonstrate our second hypothesis.

## 1.5 Contributions

This work describes our main contributions for a better understanding of graph relations through efficient processing and visualization techniques – as introduced in Section 1.3.

Firstly, we proposed a visualization technique concerning the principle that graphs can be characterized using simple structures, common in many graph domains. Instead of node relations, we represented the graph using an upper level of abstraction where sets of nodes and edges are associated with chains, stars, cliques and bipartite-cores. We designed an efficient algorithm that cuts power-law graphs iteratively. It removes hubs progressively to detect connected components in the graph; at once, it classifies each connected component in a simple structure when is possible and, when not, it applies the same cut process for each one. After detection of structures, as detailed in chapter 2, our algorithm visualizes the structures using a dense-matrix layout in which we can study the distribution of structures, their size, and how they are related.

Our second contribution was a framework for processing billion-scale graphs through a bimodal block processing strategy. Our framework reduces I/O operations on secondary memory by analysis of sparsity in subregions of the graph. For processing each subregion, it chooses between two strategies, one optimized for sparse regions and other for dense ones. In consequence, our framework minimizes disk utilization increasing sequential access, what overcomes many of the critical issues in existing frameworks. Additionally, It includes a flexible programming interface to implement popular and essential graph algorithms like PageRank, Connected Component, Sparse Matrix-Vector Multiplication, eigensolver and belief-propagation.

The rest of the document is organized as follows: Chapter 2 presents our first contribution on large graph visualization; Chapter 3 presents our bimodal block processing model for large graphs using secondary memory, and Chapter 4 presents further results that were obtained in the course of this research. Chapter 5 concludes this work.

CHAPTER

# 2

# MILLION-SCALE VISUALIZATION OF GRAPHS BY MEANS OF STRUCTURE DETECTION AND DENSE MATRICES

## 2.1 Initial considerations

Large graph visualization has been a prolific research area for years, and new techniques for processing, interaction and visualization of big graphs are usually proposed to deal with sensemaking and scalability. Given a large-scale graph with millions of nodes and millions of edges, how to reveal macro patterns of interest, like cliques, bi-partite cores, stars, and chains? Furthermore, how to visualize such patterns altogether getting insights from the graph to support wise decision-making? Although there are many algorithmic and visual techniques to analyze graphs, none of the existing approaches are able to present the structural information of graphs at million scale; hence, in this chapter, we present our first contribution StructMatrix, a methodology aimed at high-scale visual inspection of graph structures with the goal of revealing macro patterns of interest.

## 2.2 Introduction

Large-scale graphs refer to graphs generated by contemporary applications in which users or entities distributed along large geographical areas – even the entire planet – create massive amounts of information; a few examples of those are social networks, recommendation networks, road nets, e-commerce, computer networks, client-product logs, and many others. Common to such graphs is the fact that they are made of recurrent simple structures (cliques, bi-partite cores, stars, and chains) that follow macro behaviors of cardinality, distribution, and relationship. Each of these three features depends on the specific domain of the graph; therefore, each of them

characterizes the way a given graph is understood.

While some features of large graphs are detected by algorithms that produce hundreds of tabular data, these features can be better noticed with the aid of visual representations. In fact, some of these features, given their large cardinality, are intelligible, in a timely manner, exclusively with visualization. Considering this approach, we propose StructMatrix, a methodology that combines a highly scalable algorithm for structure detection with a dense matrix visualization. With StructMatrix, we introduce the following contributions:

1. **Methodology:** we introduce innovative graph processing and visualization techniques to detect macro features of very large graphs;
2. **Scalability:** we show how to visually inspect graphs with magnitudes far bigger than those of previous works;
3. **Analysis:** we analyze relevant graph domains, characterizing them according to the cardinality, distribution, and relationship of their structures.

The rest of the paper presents related works in Section 3.3, the proposed methodology in Section 2.4, experimentation in Section 2.5, and conclusions in Section 2.6. Table 1 lists the symbols used in our notation.

## 2.3 Related works

### 2.3.1 Large graph visualization

There are many works about graph visualization, however, the vast majority of them is not suited for large-scale. Techniques that are based on node-link drawings cannot, at all, cope with the needs of just a few thousand edges that would not fit in the display space. Edge bundling [50] techniques are also limited since they do not scale to millions of nodes and also because they are able to present only the main connection pathways in the graph, disregarding potentially useful details. Other large-scale techniques are visual in a different sense; they present plots of calculated features of the graph instead of depicting their structural information. This is the case of Apolo [51], Pegasus [52], and OddBall [53]. There are also techniques [54] that rely on sampling to gain scalability, but this approach assumes that parts of the graph will be absent; parts that are of potential interest.

Adjacency matrices in contrast to Node-Link diagrams are the most recommended techniques for fine inspection of graphs in scalable manner [55]; this is because they can represent an edge for each pixel in the display. However, even with one edge per pixel, one can visualize roughly a few million edges. Works Matrix Zoom[56] and ZAME[14] extend the one-edge-per-pixel approach by merging nodes and edges through clustering algorithms, creating an adjacency matrix where each position represents a set of edges on a hierarchical aggregation.

The main challenge of using clustering techniques is to find an aggregation algorithm that produces a hierarchy that is meaningful to the user. There are also matrix visualization layouts as MatLink [57] and NodeTrix [58] combining Node-Link and adjacency representations to increase readability and scalability, but those approaches are not enough to visualize large-scale graphs.

Net-Ray [30] is another technique working at large scale; it plots the original adjacency matrix of one large graph in the much smaller display space using a simple projection: the original matrix is scaled down by means of straight proportion. This approach causes many edges to be mapped to one same pixel; this is used to generate a heat map that informs the user of how many edges are in a certain position of the dense matrix.

In this work, we extend the approach of adjacency matrices, as proposed by Net-Ray, improving its scalability and also its ability to represent data. In our methodology, we introduce two main improvements: (1) our adjacency matrix is not based on the classic node-to-node representation; we first condense the graph as a collection of smaller structures, defining a structure-to-structure representation that enhances scalability as more information is represented and less compression of the adjacency matrix is necessary; and (2) our projection is not a static image but rather an interactive plotting from which different resolutions can be extracted, including the adjacency matrix with no overlapping – of course, considering only parts of the matrix that fit in the display.

### 2.3.2 Structure detection

The principle of StructMatrix is that graphs are made of simple structures that appear recurrently in any graph domain. These structures include cliques, bipartite cores, stars, and chains that we want to identify. Therefore, a given network can be represented in an upper level of abstraction; instead of nodes, we use sets of nodes and edges that correspond to substructures. The motivation here is that analysts cannot grasp intelligible meaning out of huge network structures; meanwhile, a few simple substructures are easily understood and often meaningful. Moreover, analyzing the distribution of substructures, instead of the distribution of single nodes, might reveal macro aspects of a given network.

**Partitioning (shattering) algorithms**

StructMatrix, hence, depends on a partitioning (shattering) algorithm to work. Many algorithms can solve this problem, like Cross-associations [59], Eigenspokes [60], and METIS [61], and VoG [62]. We verified that VoG overcomes the others in detecting simple recurrent structures considering a limited well-known set.

Vog relies on the technique introduced by graph compression algorithm Slash-Burn by Kang and Faloutsos [63]. The idea of Slash-Burn is that, in contrast to random graphs or

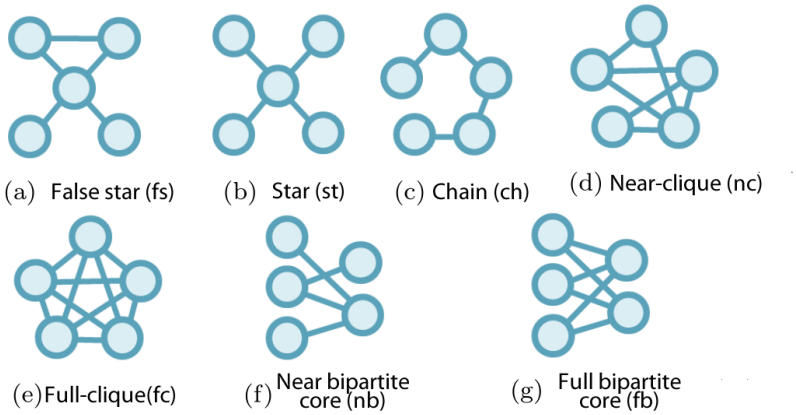



Figure 1 – The vocabulary of graph structures considered in our methodology. From (a) to (g), illustrative examples of the patterns that we consider; we process variations on the number of nodes and edges of such patterns.

lattices, the degree distribution of real-world networks obeys to power laws; in such graphs, a few nodes have a very high degree, while the majority of the nodes have low degree. Kang and Faloutsos also demonstrated that large networks are easily shattered by an ordered "removal" of the hub nodes. In fact, after each removal, a small set of disconnected components (satellites) appear, while the majority of the nodes still belong to the giant connected component. That is, the disconnected components were connected to the network only by the hub that was removed and, by progressively removing the hubs, the entire graph is scanned part by part. Interestingly, the small components that appear determine a partitioning of the network that is more coherent than cut-based approaches [64]. The technique works for any power-law graph without domain-specific knowledge or specific ordering of the nodes.

For the sake of completeness and performance, we designed a new algorithm that, following the Slash-Burn technique, extends algorithm Vog with parallelism, optimizations, and an extended vocabulary of structures, as detailed in Section 2.4.2. Our results demonstrated better performance while considering a larger set of structures.

## 2.4 Proposed method: StructMatrix

As we mentioned before, *StructMatrix* draws an adjacency matrix in which each line/column is a structure, not a single node; besides that, it uses a projection-based technique to "squeeze" the edges of the graph in the available display space, together with a heat mapping to inform the user of how big are the structures of the graph. In the following, we formally present the technique.

### 2.4.1 Overview of the graph condensation approach

For this work, we use a vocabulary of structures that extends those of former works; it considers seven well-known structures – see Figure 1 – found in the graph mining literature: false stars (*fs*), stars (*st*), chains (*ch*), *near* and *full* cliques (*nc*, *fc*), *near* and *full* bi-partite cores (*nb*, *fb*). Shortly, we define the vocabulary of structures as $\psi = \{fs, st, ch, nc, fc, nb, fb\}$.

| Notation | Description |
|---|---|
| $G(V,E)$ | graph with V vertices and E edges |
| $S, S_x$ | structure-set |
| $n$, $\|S\|$ | cardinality of $S$ |
| $M, m_{x,y}$ | StructMatrix |
| $fc$, $nc$ | full and near clique resp. |
| $fb$, $nb$ | full and near bipartite core resp. |
| $st$, $fs$, $ch$ | star, false star and chain resp. |
| $\psi$ | vocabulary (set) of structures |
| $D(s_i, s_j)$ | Number of edges between structure instances $s_i$ and $s_j$ |

Table 1 – Description of the major symbols used in this work.

*False* stars are structures similar to stars (a central node surrounded by satellites), but whose satellites have edges to other nodes, indicating that the star may be only a substructure of a bigger structure – see Figure 1. A near-clique or $\varepsilon$*-near clique* is a structure with $1-\varepsilon$ ($0 < \varepsilon < 1$) percent of the edges that a similar full clique would have; the same holds for near bipartite cores. In our case, we are considering $\varepsilon = 0.2$ so that a structure is considered *near clique* or *near* bipartite core, if it has at least 80 percent of the edges of the corresponding full structure.

The rationale behind the set of structures $\psi$ is that (a) cliques correspond to strongly connected sets of individuals in which everyone is related to everyone else; cliques indicate communities, closed groups, or mutual-collaboration societies, for instance. (b) Chains correspond to sequences of phenomena/events like those of "spread the word", according to which one individual passes his experience/feeling/impression/contact with someone else, and so on, and so forth; chains indicate special paths, viral behavior, or hierarchical processes. (c) Bipartite cores correspond to sets of individuals with specific features, but with complementary interaction; bipartite cores indicate the relationship between professors and students, customers and products, clients and servers, to name a few. And, (d) stars correspond to special individuals highly connected to many others; stars indicate hub behavior, authoritative sites, intersecting paths, and many other patterns.

Considering these motivations, our algorithm condenses the graph in a dense adjacency matrix. To do so, it produces a set with the instances of structures in $\psi$ that were found in the graph; this set of instances contains the same information as that of the original graph but with vertices and edges grouped as structures. Beyond that, the algorithm detects the edges in between the structures, so that it becomes possible to build a condensed adjacency matrix that informs which structure is connected to each other structure.

### 2.4.2 StructMatrix algorithm

As mentioned earlier, our algorithm is based on a high-degree ordered removal of hub nodes from the graph; the goal is to accomplish an efficient shattering of the graph, as introduced in Section 2.3.2. As we describe in Algorithm 1, our process relies on a queue, $\Phi$, which contains the unprocessed connected components (initially the whole graph), and a set $\Gamma$ that contains the discovered structures. In line 4, we explore the fact that the problem is straight parallelizable by triggering threads that will process each connected component in queue $\Phi$. In the process, we proceed with the ordered removal of hubs – see line 5, which produces a new set of connected components. With each connected component, we proceed by detecting a structure instance in line 7, or else, pushing it for processing in line 10. The detection of structures and the identification of their respective types occur according to Algorithm 2, which uses edge arithmetic to characterize each kind of structure.

**Algorithm 1** StructMatrix algorithm

**Require:** Graph $G = (E,V)$
**Ensure:** Array $\Gamma$ containing the structures found in $G$
1: Let be queue $\Phi = \{G\}$ and set $\Gamma = \{\}$
2: **while** $\Phi$ is not empty **do**
3: $H$ =Pop($\Phi$) /*Extract the first item from queue $\Phi$*/
4: SUBFUNCTION Thread($H$) BEGIN /*In parallel*/
5: $H'$ = "$H$ without the 1% nodes with highest degree"
6: **for** each connected component $cc \in H'$ **do**
7: **if** $cc \in \psi$ using Algorithm 2 **then**
8: Add($\Gamma$, cc)
9: **else**
10: Push($\Phi$, cc)
11: **end if**
12: **end for**
13: END Thread($H$)
14: **end while**

**Algorithm 2** Structure classification

**Require:** Subgraph $H = (E,V)$; $n = |V|$ and $m = |E|$
1: **if** m = $\frac{n(n-1)}{2}$ **then return** *fc*
2: **else if** $m > (1-\varepsilon) * \frac{n(n-1)}{2}$ **then return** *nc*
3: **else if** $m < \frac{n^2}{4}$ and $H = (E, V_a \cup V_b)$ is bipartite **then**
4: **if** $m = |V_a| * |V_b|$ **then return** *fb*
5: **else if** $m > (1-\varepsilon) * |V_a| * |V_b|$ **then return** *nb*
6: **else if** $|V_a| = 1$ or $|V_b| = 1$ **then return** *st*
7: **else if** $m = n-1$ **then return** *ch*
8: **end if**
9: **end ifreturn** undefined structure

The StructMatrix algorithm, different from former works, maximizes the identification

of structures rather than favoring optimum compression; it uses parallelism for improved performance; and considers a larger set of structures. In Section 2.5, we demonstrate these aspects through experimentation.

### 2.4.3 Adjacency Matrix Layout

A graph $G = \langle V, E \rangle$ with $V$ vertices and $E$ edges can be expressed as a set of structural instances $S = \{s_0, s_2, \ldots, s_{|S|-1}\}$, where $s_i$ is a subgraph of $G$ that is categorized – see Figure 1 and Table 1 – according to the function $type(s) : S \rightarrow \psi$. To create the adjacency matrix of structures, first we identify the set $S$ of structures in the graph and categorize each one. Following, we define $n = |S|$ to refer to the cardinality of $S$.

As depicted in Figure 2, each type of structure defines a partition in the matrix, both horizontally and vertically, determining subregions in the visualization matrix. In this matrix, a given structure instance corresponds to a horizontal and to a vertical line (w.r.t. the subregions) in which each pixel represents the presence of edges (one or more) between this structure and the others in the matrix. Therefore, the matrix is symmetric and supports the representation of relationships (edges) between all kinds of structure types. Formally, the elements $m_{i,j}$ of a StructMatrix $M_{n\times n}$, $0 < i < (n-1)$ and $0 < j < (n-1)$ are given by:

$$m_{i,j} = \begin{cases} 1, \; if \; D(s_i, s_j) > 0; \\ 0 \; otherwise. \end{cases} \tag{2.1}$$

where $D : S \times S \rightarrow \mathbb{N}$ is a function that returns the number of edges between two given structure instances. For quick reference, please refer to Table 1.

In this work, we focus on large-scale graphs whose corresponding adjacency matrices do not fit in the display. This problem is lessened when we plot the structures-structures matrix, instead of the nodes-nodes matrix. However, due to the magnitude of the graphs, the problem persists. We treat this issue with a density-based visualization for each subregion formed by two types of structures $(\psi_i, \psi_j), \psi_i \in \psi$ and $\psi_j \in \psi$ – for example, $(fs, fs), (fs, st), ...$, and so on. In each subregion, we map each point of the original matrix according to a straight proportion. We map the lower, left boundary point $(x_{min}, y_{min})$ to the center of the lower, left boundary pixel; and the upper, right boundary point $(x_{max}, y_{max})$ to the center of the upper, right boundary pixel. The remaining points are mapped as $(x, y) \rightarrow (\rho_x, \rho_y)$ for:

$$\begin{aligned} \rho_x &= R(\psi_i, \psi_j) + \left\lceil (Res_x - 1)\tfrac{x - x_{min}}{x_{max} - x_{min}} + \tfrac{1}{2} \right\rceil \\ \rho_y &= R(\psi_i, \psi_j) + \left\lceil (Rex_y - 1)\tfrac{y - y_{min}}{y_{max} - y_{min}} + \tfrac{1}{2} \right\rceil \end{aligned} \tag{2.2}$$

where $R : \psi \times \psi \rightarrow \mathbb{N}$ is a function that returns the offset (left boundary) in pixels of the region $(\psi_i, \psi_j)$ and $Res_x, Res_y$ are the target resolutions. The more resolution, the more details are

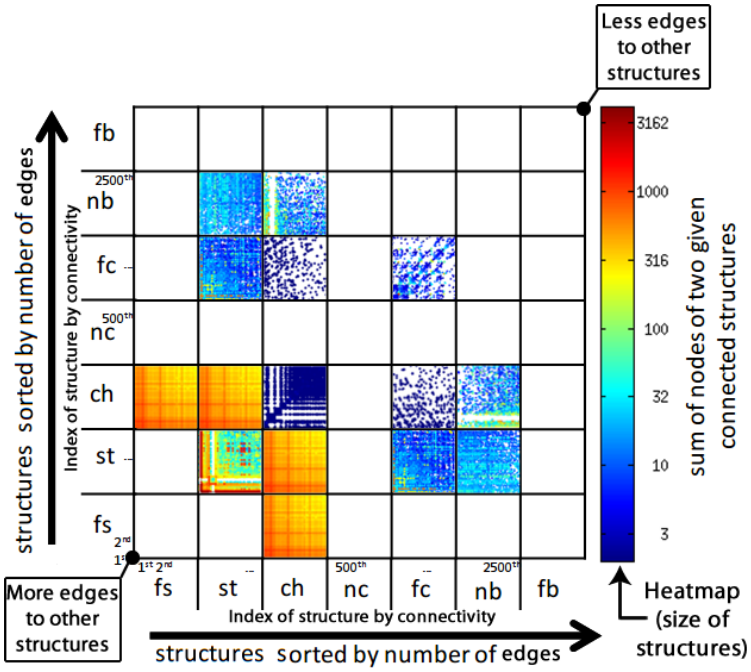



Figure 2 – Adjacency Matrix layout.

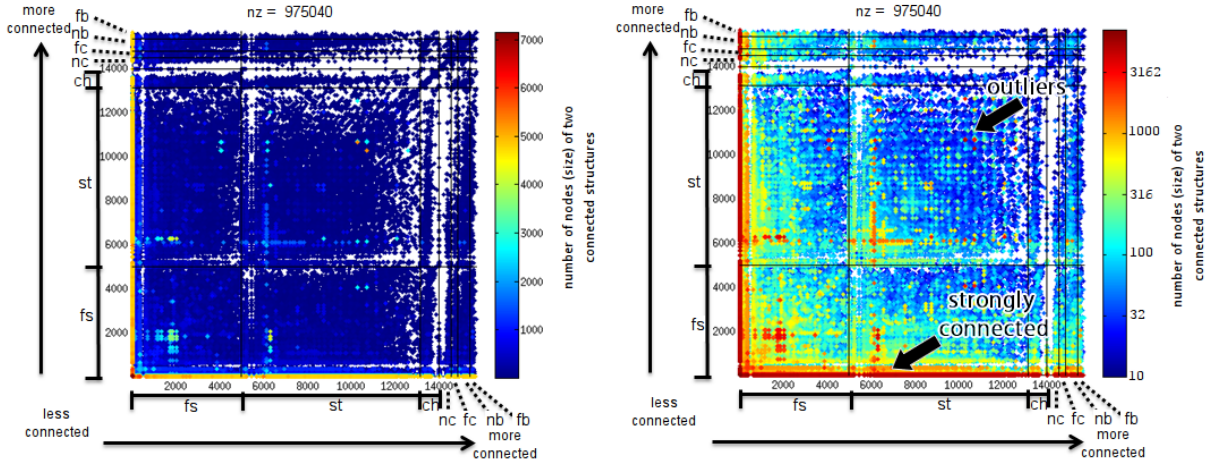
presented, these parameters allow for interactive grasping of details.

Each set of edges connecting two given structures is then mapped to the respective subregion of the visualization where the structures' types cross. Inside each structure subregion we add an extra information by ordering the structure instances according to the number of edges that they have to other structures; that is, by $\sum_{i=0}^{|S|-1} D(s, s_i)$.

Therefore, the structures with the largest number of edges to other structures appear first – more at the bottom left, less at the top right, of each subregion as explained in Figure 2.

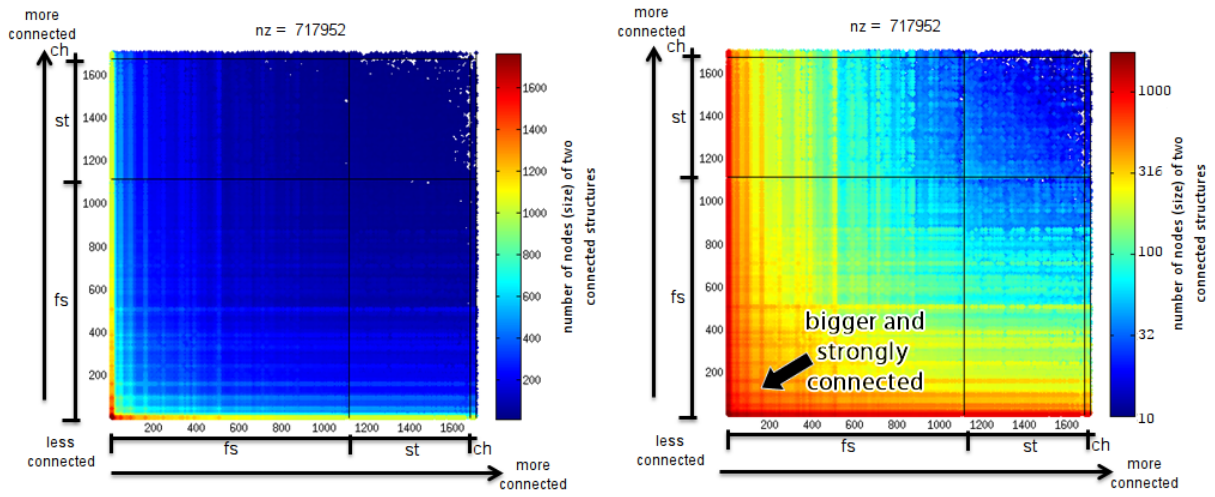
In the visualization, each horizontal/vertical line (w.r.t. the subregions) corresponds to a few hundred or thousand structure instances; and each pixel corresponds to a few hundred or thousand edges. We deal with that by not plotting the matrix as a static image, but as a dynamic plot that adapts to the available space; hence, it is possible to select specific areas of the matrix and see more details of the edges. It is possible to regain details until reaching parts the original plot, when all the edges are visible.

We plot one last information using color to express the sum of nodes of two given connected structures. We use a color map in which the smaller number of nodes is indicated with bluish colors and the bigger number of nodes is indicated with reddish colors. In addition, we use the same information as used for color encoding to determine the order of plotting: first we plot the edges of the smaller structures (according to the number of nodes), and then the edges of the bigger structures. This procedure assures that the hotter edges will be over the cooler ones, and that the interesting (bigger) structures will be spotted easier. At this point the elements $m_{i,j}$




(a) Normal scale. (b) Log scale.

Figure 3 – StructMatrix in the WWW-barabasi graph with colors displaying the sum of the sizes of two connected structures; in the graph, stars refer to websites with links to other websites.




(c) Normal scale. (d) Log scale.

Figure 4 – StructMatrix in the Wikipedia-vote graph with values displaying the sum of the sizes of two connected structures; in this graph, stars refer to users who got/gave votes from/to other users.

of a StructMatrix $M_{n\times n}$, $0 < i < (n-1)$ and $0 < j < (n-1)$ are given by:

$$m_{i,j} = \begin{cases} C(NNodes(s_i) + NNodes(s_j)), \\ if\ D(s_i, s_j) > 0; \\ \\ 0\ otherwise. \end{cases} \tag{2.3}$$

where $NNodes : S \to \mathbb{N}$ is a function that returns the number of nodes of a given structure instance; and $C : \mathbb{N} \to [0.0, 1.0]$ is a function that returns a continuous value between 0.0 (cool blue for smaller structures) and 1.0 (hot red for bigger structures) according to the sum of the number of nodes in the two connected structures. In our visualization, we map the function $C$ to a log scale and then we apply a linear color scale to the data.

| Graph | fs | st | ch | nc | fc | nb | fb |
|---|---|---|---|---|---|---|---|
| DBLP | 122,983 (76%) | 7,585(5%) | 3,096(2%) | 2,656(2%) | 24,551(15%) | 14(<1%) | - |
| WWW-barabasi | 4,957(32%) | 8,146(52%) | 851(5%) | 541(3%) | 283(2%) | 556(4%) | 318(2%) |
| cit-HepPh | 11,449(79%) | 1,948(13%) | 840(6%) | 120(1%) | 44 (4<1%) | 35(<1%) | 43(<1%) |
| Wikipedia-vote | 1,112(65%) | 564(33%) | 29 (2%) | - | - | 1(<1%) | - |
| Epinions | 4,518(52%) | 2,725(31%) | 1,247(14%) | 28 (%) | 21(%) | 150(2%) | 3(<1%) |
| Roadnet PA | 11,825(23%) | 22,934(45%) | 13,748(27%) | - | - | 2,668(5%) | - |
| Roadnet CA | 24,193(27%) | 34,781(39%) | 26,236(29%) | - | - | 3,763(4%) | - |
| Roadnet TX | 15,595(25%) | 27,094(43%) | 17,457(28%) | - | - | 2,468(4%) | - |

Table 3 – Structures found in the datasets considering a minimum size of 5 nodes.

## 2.5 Experiments

Table 2 describes the graphs we use in the experiments.

| Name | Nodes | Edges | Description |
|---|---|---|---|
| DBLP | 1,366,099 | 5,716,654 | Collaboration network |
| Roads of PA | 1,088,092 | 1,541,898 | Road net of Pennsylvania |
| Roads of CA | 1,965,206 | 2,766,607 | Road net of California |
| Roads of TX | 1,379,917 | 1,921,660 | Road net of Texas |
| WWW-barabasi | 325,729 | 1,090,108 | WWW in nd.edu |
| Epinions | 75,879 | 405,740 | Who-trusts-whom network |
| cit-HepPh | 34,546 | 420,877 | Co-citation network |
| Wiki-vote | 7,115 | 100,762 | Wikipedia votes |

Table 2 – Description of the graphs used in our experiments.

### 2.5.1 Graph condensations

Table 3 shows the condensation results of the structure detection algorithm over each dataset, already considering the extended vocabulary and structures with minimum size of 5 nodes – less than 5 nodes could prevent to tell apart the structure types. The columns of the table indicate the percentage of each structure identified by the algorithm. For all the datasets, the false star was the most common structure; the second most common structure was the star, and then the chain, especially observed in the road networks. The improvement of the visual scalability of *StructMatrix*, compared to former work Net-Ray, is as big as the amount of information that is "saved" when a graph is modeled as a structure-to-structure adjacency matrix, instead of a node-to-node matrix.

### 2.5.2 Scalability

In order to test the processing scalability of *StructMatrix*, we used a breadth-first search over the DBLP dataset to induce subgraphs of different sizes – we created graphs ranging from 50K edges up to 1.000K edges. For the scalability experiment, we used a contemporary commercial desktop (Intel i7 with 8 GB RAM). We compared the performance between *VoG* and *StructMatrix* to detect simple recurrent structures from a limited well-known set. Figure 5 shows that StructMatrix and

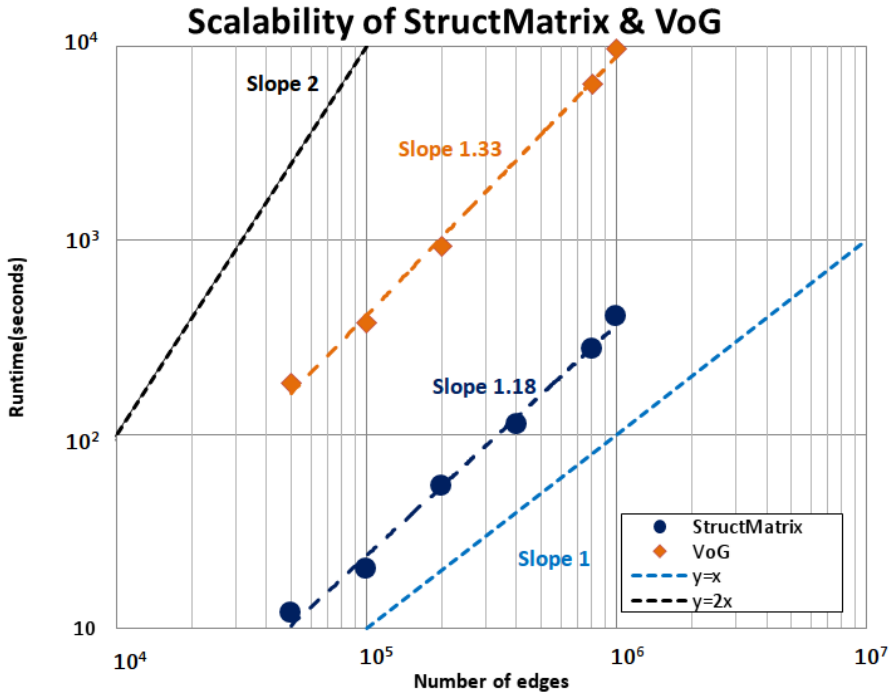



Figure 5 – Scalability of the *StructMatrix* and VoG techniques; although VoG is near-linear to the graph edges, StructMatrix overcomes VoG for all the graph sizes.

VoG are near-linear on the number of edges of the input graph, however StructMatrix overcomes VoG for all the graph sizes.

### 2.5.3 WWW and Wikipedia

In Figures 3 and 4, one can see the results of StructMatrix for graphs WWW-barabasi (325,729 nodes and 1,090,108 edges) and Wikipedia-vote (7,115 nodes and 100,762 edges) condensed as described in Table 3. For graph WWW-barabasi, Figure 3a shows the StructMatrix with linear color encoding, and Figure 3b shows the StructMatrix with logarithmic color encoding. For the Wikipedia-vote graph, the same visualizations are presented in Figures 3c and 3d. We observe the following factors in the visualizations:

- the share of structures: WWW-barabasi presents a clear majority of stars, followed by false stars, and chains, while the Wikipedia-vote presents a majority of false stars, followed by stars, and chains; in both cases, stars strongly characterize each domain, as expected in websites and in elections;
- the presence of outliers in WWW-barabasi, spotted in red; and the presence of structures globally and strongly connected in Wikipedia-vote, depicted as reddish lines across the visualization;
- the notion that the bigger the structures, the more connected they are – reddish (the bigger) structures concentrate on the left (the more connected), especially perceived in Wikipedia-vote;

- the effect of the logarithmic color scale; its use results in a clearer discrimination of the magnitudes of the color-mapped values, what helps to perceive the distribution of the values; more skewed in WWW and more uniform in Wikipedia.

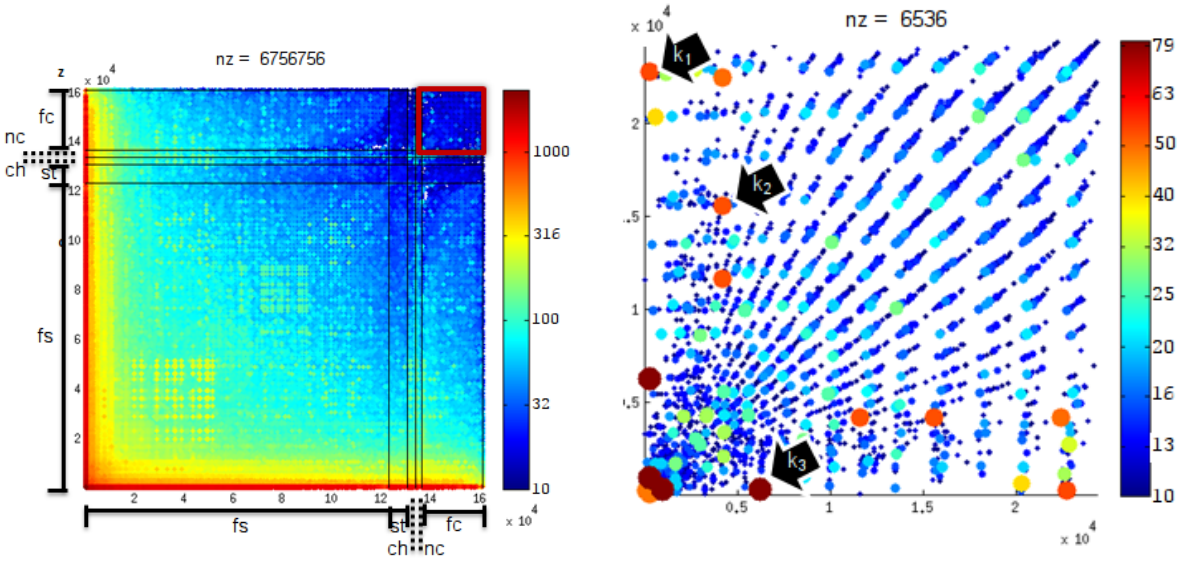
The stars and false stars of the WWW graph in Figure 3b refer to sites with multiple pages and many out-links – bigger sites are reddish, more connected sites to the left. The visualization is able to indicate the big stars (sites) that are well-connected to other sites (reddish lines), and also the big sites that demand more connectivity – reddish isolated pixels. The chains indicate site-to-site paths of possibly related semantics, an occurrence not so rare for the WWW domain. There is also a set of reasonably small, interconnected sites that connect only with each other and not with the others – these sites determine blank lines in the visualization and their sizes are noticeable in dark blue at the bottom-left corner of the star-to-star subregion. Such sites should be considered as outliers because, although strongly connected, they limit their connectivity to a specific set of sites.

While the Wikipedia graph is mainly composed of stars, just like the WWW graph, the Wikipedia graph is quite different. Its structures are more interconnected defining a highly populated matrix. That means that users (contributors) who got many votes to be elected as administrators in Wikipedia, also voted in many other users. The sizes of the structures, indicated by color, reveal the most voted users, positioned at the bottom-left corner – the color pretty much corresponds to the results of the elections: of the 2,794 users, only 1,235 users had enough votes to be elected administrators (nearly 50% of the reddish area of the matrix). There are also a few chains, most of them connected to stars (users), especially the most voted ones – it becomes evident that the most voted users also voted on the most voted users. This is possibly because, in Wikipedia, the most active contributors are aware of each other.

### 2.5.4 Road networks

On the road networks, if we consider the stars segment (“st”), each structure corresponds to a city (the intersecting center of the star); therefore, the horizontal/vertical lines of pixels correspond to the more important cities that act as hubs in the road system. Its *StructMatrix* visualization – Figure 7 – showed an interesting pattern for all the three road datasets: in the figure, one can see that the relationships between the road structures is more probable in structures with similar connectivity. This fact is observable in the curves (diagonal lines of pixels) that occur in the visualization – remember that the structures are first ordered by type into segments, and then by their connectivity (more connected first) in each segment.

Another interesting fact is the presence of some structures heavily connected to nearly all the other structures; these structures define horizontal lines of pixels in the visualization and, due to symmetry, they also define vertical lines of pixels. The same patterns were observed for roadnets from California, Texas, and Pennsylvania. According to the visualizations, roads are characterized by three patterns:




(a) All types of structures. (b) Only the *fc-fc* sub region with details.

Figure 6 – DBLP Zooming on the *full* clique section.

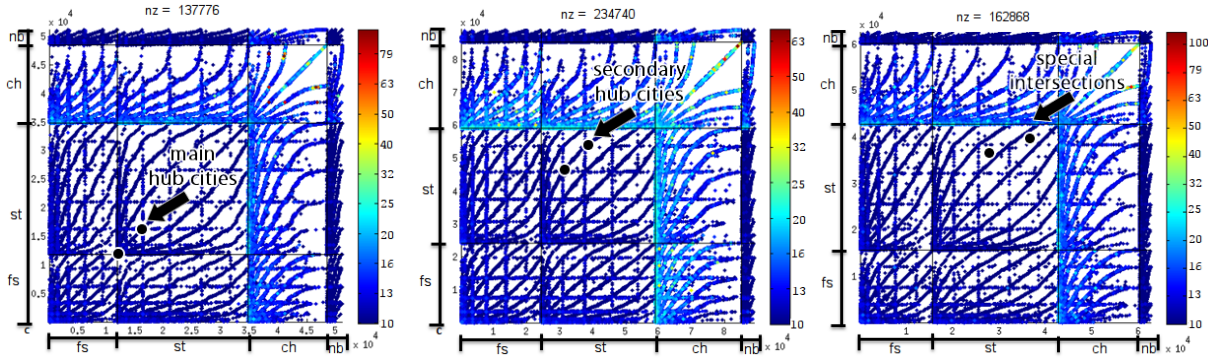



Figure 7 – StructMatrix with colors in log scale indicating the size of the structures interconnected in the road networks of Pennsylvania (PA), California (CA) and Texas(TX). Again, stars appear as the major structure type; in this case they correspond to cities or to major intersections.

1. cities that connect to most of the other cities acting as interconnecting centers in the road structure; these cities are of different importance and occur in small number – around 6 for each state that we studied;
2. there is a hierarchical structure dictated by the connectivity (importance) of the cities; in this hierarchy, the connections tend to occur between cities with similar connectivity; one consequence of this fact is that going from one city to some other city may require one to first "ascend" to a more connected city; actually, for this domain, the lines of pixels in the visualization correspond to paths between cities, passing through other cities – the bigger the inclination of the line, the shorter the path (the diagonal is the longest path);
3. road connections that are out of the hierarchical pattern – the ones that do not pertain to any line of pixels; such connections refer to special roads that, possibly, were built on specific demands, possibly not obeying to the general guidelines for road construction.

From these visualizations and patterns, we notice that the StructMatrix visualization is a quick way (seconds) to represent the structure of graphs on the order of million-nodes (intersections) and million-edges (roads). For the specific domain of roads, the visualization spots the more important cities, the hierarchy structure, outlier roads that should be inspected closer, and even, the adequacy of the roads' inter connectivity. This last issue, for example, may indicate where there should be more roads so as to reduce the pathway between cities.

### 2.5.5 DBLP

In the *StructMatrix* of the DBLP co-authoring graph – see Figure 6a – it is possible to see a huge number of false stars. This fact reflects the nature of DBLP, in which works are done by advisors who orient multiple students along time; these students in turn connect to other students defining new stars and so on. A minority of authors, as seen in the matrix, concerns authors whose students do not interact with other students defining stars properly said. The presence of full cliques (*fc*) is of great interest; sets of authors that have co-authorship with every other author. Full cliques are expected in the specific domain of DBLP because every paper defines a full clique among its authors – this is not true for all clique structures, but for most of them.

In Figure 6b, we can see the full clique-to-full clique region in more details and with some highlights indicated by arrows. The Figure highlights some notorious cliques: $k_1$ refers to the publication with title "*A 130.7mm 2-layer 32Gb ReRAM memory device in 24nm technology*" with 47 authors; $k_2$ refers to paper "*PRE-EARTHQUAKES, an FP7 project for integrating observations and knowledge on earthquake precursors: Preliminary results and strategy*" with 45 authors; and $k_3$ refers to paper "*The Biomolecular Interaction Network Database and related tools 2005 update*" with 75 authors. These specific structures were noticed due to their colors, which indicate large sizes. Structures $k_1$ and $k_3$, although large, are mostly isolated since they do not connect to other structures; $k_2$, on the other hand, defines a line of pixels (vertical and horizontal) of similarly colored dots, indicating that it has connections to other cliques.

## 2.6 Conclusions

We focused on the problem of visualizing graphs so big that their adjacency matrices demand much more pixels than what is available in regular displays. We advocate that these graphs deserve macro analysis; that is, analysis that reveal the behavior of thousands of nodes altogether, and not of specific nodes, as that would not make sense for such magnitudes. In this sense, we provide a visualization methodology that benefits from a graph analytical technique. Our contributions are:

- **Visualization technique:** we introduce a processing and visualization methodology that puts together algorithmic techniques and design in order to reach large-scale visualizations;

- **Analytical scalability:** our technique extends the most scalable technique found in the literature; plus, it is engineered to plot millions of edges in a matter of seconds;
- **Practical analysis:** we show that large-scale graphs have well-defined behaviors concerning the distribution of structures, their size, and how they are related one to each other; finally, using a standard laptop, our techniques allowed us to experiment in real, large-scale graphs coming from domains of high impact, i.e., WWW, Wikipedia, Roadnet, and DBLP.

Our approach can provide interesting insights on real-life graphs of several domains answering to the demand that has emerged in the last years. By converting the graph's properties into a visual plot, one can quickly see details that algorithmic approaches either would not detect, or that would be hidden in thousand-lines tabular data.

## 2.7 Final considerations

In this chapter, we presented StructMatrix, an innovative approach that combines algorithmic structure detection and adjacency matrix visualization to present cardinality, distribution, and relationship of the structures found in a given graph. We performed experiments in real, million-scale graphs with up to millions of nodes and over 5 million edges. StructMatrix revealed that graphs of high relevance (e.g., Web, Wikipedia and DBLP) have characterizations that reflect the nature of their corresponding domains; our findings have not been seen in the literature so far. We expect that our technique will bring deeper insights into large graph mining, leveraging their use for decision making.

CHAPTER

3

# M-FLASH: FAST BILLION-SCALE GRAPH COMPUTATION USING A BIMODAL BLOCK PROCESSING MODEL

## 3.1 Initial considerations

Recent graph computation approaches such as GraphChi, X-Stream, TurboGraph and MMap demonstrated that a single PC can perform efficient computation on billion-scale graphs. While they use different techniques to achieve scalability through optimizing I/O operations, such optimization often does not fully exploit the capabilities of modern hard drives. In this chapter, we present our novel and scalable graph computation framework called M-Flash, which uses a new, bimodal block processing strategy (*BBP*) to boost computation speed by minimizing I/O cost.

## 3.2 Introduction

Large graphs with **billions** of nodes and edges are increasingly common in many domains and applications, such as in studies of social networks, transportation route networks, citation networks, and many others. Distributed frameworks have become popular choices for analyzing these large graphs (e.g., GraphLab [41], PEGASUS [65] and Pregel [9]). However, distributed approaches may not always be the best option, because they can be expensive to build [45], hard to maintain and optimize.

These potential challenges prompted researchers to create single-machine, billion-scale graph computation frameworks that are well-suited to essential graph algorithms, such as eigensolver, PageRank, connected components and many others. Examples are GraphChi [45] and TurboGraph [48]. Frameworks in this category define sophisticated processing schemes

to overcome challenges induced by limited main memory and poor locality of memory access observed in many graph algorithms [66]. For example, most frameworks use an iterative, vertex-centric programming model to implement algorithms: in each iteration, a *scatter* step first propagates the data or information associated with vertices (e.g., node degrees) to their neighbors, followed by a *gather* step, where a vertex accumulates incoming updates from its neighbors to recalculate its own vertex data.

Recently, X-Stream [11] introduced a related edge-centric, scatter-gather processing scheme that achieved better performance over the vertex-centric approaches, by favoring sequential disk access over unordered data, instead of favoring random access over ordered and indexed data (as it occurs in most other approaches). When studying this and other approaches [67][45], we noticed that despite their sophisticated schemes and novel programming models, they often do not optimize for disk operations, which is the core of performance in graph processing frameworks. For example, reading or writing to disk is often performed at a lower speed than the disk supports; or, reading from disk is commonly executed more times than it is necessary, what could be avoided. In the context of **single-node**, **billion-scale** graph processing frameworks, we present **M-Flash**, a novel scalable framework that overcomes many of the critical issues of the existing approaches. M-Flash outperforms the state-of-the-art approaches in large graph computation, being many times faster than the others. More specifically, our contributions include:

1. M-Flash Framework & Methodology: we propose the novel M-Flash framework that achieves fast and scalable graph computation via our new bimodal block model that significantly boosts computation speed and reduces disk accesses by dividing a graph and its node data into blocks (dense and sparse), thus minimizing the cost of I/O. Complete source-code of M-Flash is released in open source: `https://github.com/M-Flash`.

2. **Programming Model:** M-Flash provides a flexible, and deliberately simple programming model, made possible by our new bimodal block processing strategy. We demonstrate how popular, essential graph algorithms may be easily implemented (e.g., PageRank, connected components, the *first* single-machine eigensolver over billion-node graphs, etc.), and how a number of others can be supported.

3. **Extensive Experimental Evaluation:** we compared M-Flash with state-of-the-art frameworks using large real graphs, the largest one having 6.6 billion edges (YahooWeb [7]). M-Flash was consistently and significantly faster than GraphChi [45], X-Stream [11], TurboGraph [48] and MMap [67] across all graph sizes. And it sustained high speed even when memory was severely constrained (e.g., 6.4X faster than X-Stream, when using 4GB of RAM).

## 3.3 Related work

A typical approach to scalable graph processing is to develop a distributed framework. This is the case of PEGASUS [65], Apache Giraph (`http://giraph.apache.org.`), Powergraph [41], and Pregel [9]. Differently, in this work, we aim to scale up by maximizing what a single machine can do, which is considerably cheaper and easier to manage. Single-node processing solutions have recently reached a comparative performance with distributed systems for similar tasks [46].

Among the existing works designed for single-node processing, some of them are restricted to SSDs. These works rely on the remarkable low-latency and improved I/O of SSDs compared to magnetic disks. This is the case of TurboGraph [48] and RASP [49], which rely on random accesses to the edges — not well supported over magnetic disks. Our proposal, M-Flash, avoids this drawback at the same time that it demonstrates better performance over TurboGraph.

GraphChi [45] was one of the first single-node approaches to avoid random disk/edge accesses, improving the performance for mechanical disks. GraphChi partitions the graph on disk into units called *shards*, requiring a preprocessing step to sort the data by source vertex. GraphChi uses a vertex-centric approach that requires a shard to fit entirely in memory, including both the vertices in the shard and all their edges (in and out). As we demonstrate, this fact makes GraphChi less efficient when compared to our work. Our M-Flash requires only a subset of the vertex data to be stored in memory.

MMap [67] introduced an interesting approach based on OS-supported mapping of disk data into memory (virtual memory). It allows graph data to be accessed as if they were stored in unlimited memory, avoiding the need to manage data buffering. This enables high performance with minimal code. Inspired by MMap, our framework uses memory-mapping when processing edge blocks, but with an improved engineering, our M-Flash consistently outperforms MMap, as we demonstrate.

Our M-Flash also draws inspiration from the edge streaming approach introduced by X-Stream's processing model [11], improving it with fewer disk writes for dense regions of the graph. Edge streaming is a sort of stream processing referring to unrestricted data flows over a bounded amount of buffering. As we demonstrate, this leads to optimized data transfer by means of less I/O and more processing per data transfer.

## 3.4 M-Flash

In this section, we first describe how a graph is represented in M-Flash (Subsection 3.4.1). Then, we detail how our block-based processing model enables fast computation while using less RAM (Subsection 3.4.2). Subsection 3.4.3 explains how graph algorithms can be implemented using M-Flash's generic programming model, taking as examples well-known, essential algorithms.

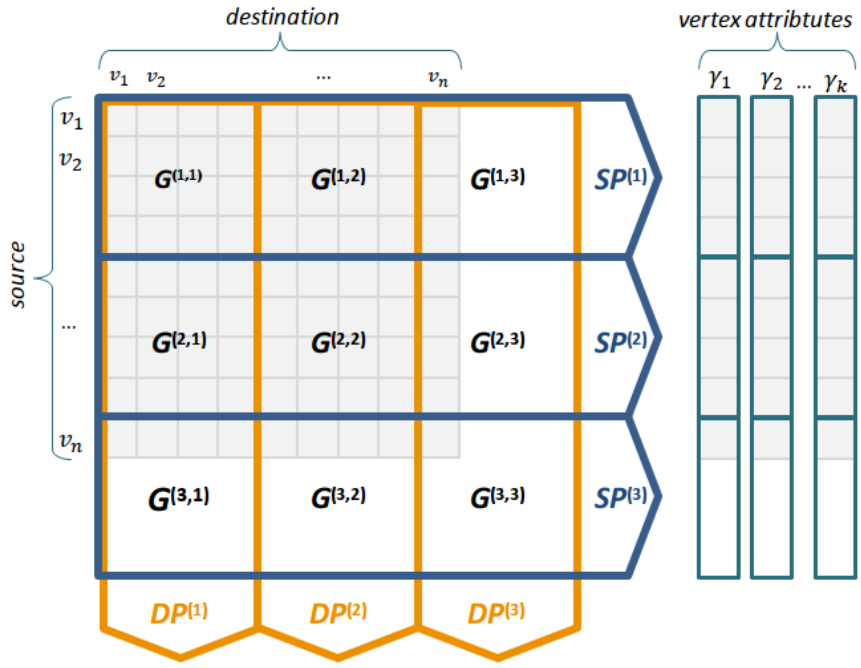



Figure 8 – Organization of edges and vertices in M-Flash. **Left (edges)**: example of a graph's adjacency matrix (in light blue color) organized in M-Flash using 3 logical intervals ($\beta = 3$); $G^{(p,q)}$ is an edge block with source vertices in interval $I^{(p)}$ and destination vertices in interval $I^{(q)}$; $SP^{(p)}$ is a *source-partition* contaning all blocks with source vertices in interval $I^{(p)}$;$DP^{(q)}$ is a *destination-partition* contaning all blocks with destination vertices in interval $I^{(q)}$. **Right (vertices):** the data of the vertices as $k$ vectors ($\gamma_1$ ... $\gamma_k$), each one divided into $\beta$ logical segments.

Finally, system design and implementation are discussed in Section 3.4.4.

The design of M-Flash considers the fact that real graphs have varying density of edges; that is, a given graph contains dense regions with more edges than other regions that are sparse. In the development of M-Flash, and through experimentation with existing works, we noticed that these dense and sparse regions could not be processed in the same way. We also noticed that this was the reason why existing works failed to achieve superior performance. To cope with this issue, we designed M-Flash to work according to two distinct processing schemes: Dense Block Processing (DBP) and Streaming Partition Processing (SPP). Hence, for full performance, M-Flash uses a theoretical I/O cost to decide the kind of processing for a given block, which can be dense or sparse. The final approach, which combines DBP and SPP, was named Bimodal Block Processing (BBP).

### 3.4.1 Graphs Representation in M-Flash

A graph in M-Flash is a directed graph $G = (V, E)$ with vertices $v \in V$ labeled with integers from 1 to $|V|$, and edges $e = (source, destination)$, $e \in E$. Each vertex has a set of attributes $\gamma = \{\gamma_1, \gamma_2, \ldots, \gamma_K\}$.

***Blocks* in M-Flash**: Given a graph $G$, we divide its vertices $V$ into $\beta$ intervals denoted by $I^{(p)}$,


Figure 9 – M-Flash's computation schedule for a graph with 3 intervals. Vertex intervals are represented by vertical (Source I) and horizontal (Destination I) vectors. Blocks are loaded into memory, and processed, in a vertical zigzag manner, indicated by the sequence of red, orange and yellow arrows. This enables the reuse of input (e.g., when going from $G^{(3,1)}$ to $G^{(3,2)}$, M-Flash reuses source node interval $I^{(3)}$), which reduces data transfer from disk to memory, boosting the speed.

where $1 \leq p \leq \beta$. Note that $I^{(p)} \cap I^{(p')} = \varnothing$ for $p \neq p'$, and $\bigcup_p I^{(p)} = V$. Thus, as shown in Figure 8, the edges are divided into $\beta^2$ *blocks*. Each block $G^{(p,q)}$ has a **source node interval** $p$ and a **destination node interval** $q$, where $1 \leq p,q \leq \beta$. In Figure 8, for example, $G^{(2,1)}$ is the block that contains edges with source vertices in the interval $I^{(2)}$ and destination vertices in the interval $I^{(1)}$. In total, we have $\beta^2$ blocks. We call this on-disk organization of the graph as **partitioning**. Since M-Flash works by alternating one entire block in memory for each running thread, the value of $\beta$ is automatically determined by equation:

$$\beta = \left\lceil \frac{2\phi T |V|}{M} \right\rceil \tag{3.1}$$

in which, $M$ is the available RAM, $|V|$ is the total number of vertices in the graph, $\phi$ is the amount of data needed to store each vertex, and $T$ is the number of threads. For example, for 1 GB RAM, a graph with 2 billion nodes, 2 threads, and 4 bytes of data per node, $\beta = \lceil (2 \times 8 \times 2 \times 2 * 10^9)/(2^{30}) \rceil = 30$, thus requiring $30^2 = 900$ blocks.

### 3.4.2 The M-Flash Processing Model

This section presents our proposed M-Flash. We first describe two strategies targeted at processing dense and sparse blocks. Next, we explain the cost-based optimization of M-Flash to take the best of them.

**Dense Block Processing (DBP)**: Figure 9 illustrates the DBP processing; notice that vertex intervals are represented by vertical (Source I) and horizontal (Destination I) vectors. After

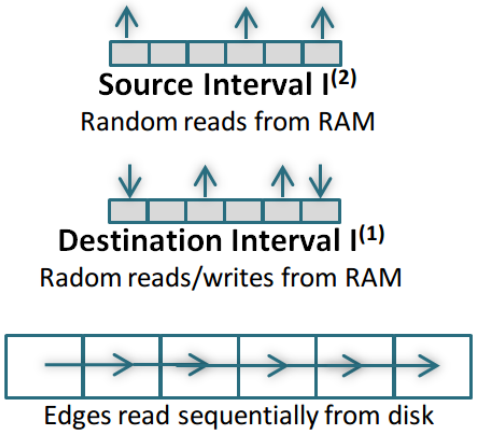



Figure 10 – Example I/O operations to process the *dense* block $G^{(2,1)}$.

partitioning the graph into *blocks*, we process them in a vertical zigzag order, as illustrated in Figure 9. There are three reasons for this order: (1) we store the computation results in the destination vertices; so, we can "pin" a destination interval (e.g., $I^{(1)}$) and process all the vertices that are sources to this destination interval (see the red vertical arrow); (2) using this order leads to fewer reads because the attributes of the destination vertices (horizontal vectors in the illustration) only need to be read once, regardless of the number of source intervals. (3) after reading all the blocks in a column, we take a "U turn" (see the orange arrow) to benefit from the fact that the data associated with the previously-read source interval is already in memory, so we can reuse that.

Within a block, besides loading the attributes of the source and destination intervals of vertices into RAM, the corresponding edges $e = \langle source, destination, edge\ properties \rangle$ are sequentially read from disk, as explained in Fig. 10. These edges, then, are processed using a user-defined function so to achieve a given desired computation. After all blocks in a column are processed, the updated attributes of the destination vertices are written to disk.

**Streaming Partition Processing (SPP)**: The performance of DBP decreases for graphs with very low density (sparse) blocks; this is because, for a given block, we have to read more data from the source intervals of vertices than from the very blocks of edges. For such situations, we designed the technique named **Streaming Partition Processing (SPP)**. The SPP processes a given graph using partitions instead of blocks. A graph *partition* can be a set of *blocks* sharing the same *source node interval* – a line in the logical partitioning, or, similarly, a set of *blocks* sharing the same *destination node interval* – a column in the logical partitioning. Formally, a *source-partition* $SP^{(p)} = \bigcup_q G^{(p,q)}$ contains all blocks with edges having source vertices in the interval $I^{(p)}$; a *destination-partition* $DP^{(q)} = \bigcup_p G^{(p,q)}$ contains all blocks with edges having destination vertices in the interval $I^{(q)}$. For example, in Figure 8, $DP^{(3)}$ is the union of the blocks $G^{(1,3)}$, $G^{(2,3)}$ and $G^{(3,3)}$.

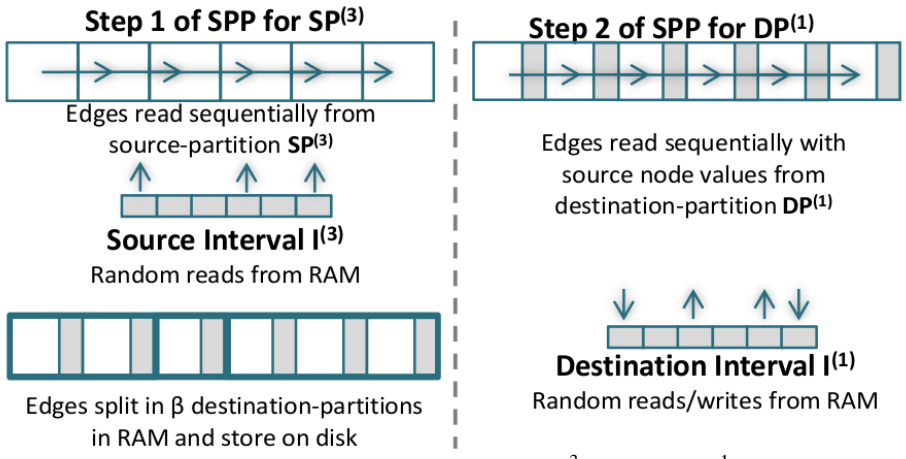



Figure 11 – Example I/O operations for step 1 of *source-partition* $SP^3$. Edges of $SP^1$ are combined with their source vertex values. Next, the edges are divided by $\beta$ *destination-partitions* in memory; and finally, edges are written to disk. On Step 2 ,*destination-partitions* are processed sequentially. Example I/O operations for step 2 of *destination-partition* $DP^{(1)}$.

For processing the graph using *SPP*, we divide the graph in $\beta$ *source-partitions*. Then, we process partitions using a two-step approach (see Fig. 11). In the first step for each *source-partition*, we load vertex values of the interval $I^{(p)}$; next, we read edges of the partition $SP^{(p)}$ sequentially from disk, storing in a temporal buffer edges together with their in-vertex values until the buffer is full. Later, we shuffle the buffer in-place, grouping edges by *destination-partition*. Finally, we store to disk edges in $\beta$ different files, one by *destination-partition*. After we process the $\beta$ *source-partitions*, we get $\beta$ *destination-partitions* containing edges with their source values. In the second step for each *destination-partition*, we initialize vertex values of interval $I^{(q)}$; next, we read edges sequentially, processing their values through a user-defined function. Finally, we store vertex values of interval $I^{(q)}$ on disk. The *SPP* model is an improvement of the edge streaming approach used in X-Stream [11]; different from former proposals, SSP uses only one buffer to shuffle edges, reducing memory requirements.

**Bimodal Block Processing (BBP)**: Schemes *DBP* and *SPP* improve the graph performance in opposite directions.

- **how can we decide which processing scheme to use when we are given a graph block to process?**

To answer this question, we propose to join DBP and SSP into a single scheme – the Bimodal Block Processing (BBP). The combined scheme uses the theoretical I/O cost model proposed by Aggarwal and Vitter [68] to decide for *SBP* or *SPP*. In this model, the I/O cost for an algorithm is equal to the number of blocks with size $B$ transferred between disk and memory plus the number of non-sequential seeks.

For processing a graph $G$, *DBP* performs the following operations over disk: one read of the edges, $\beta$ reads of the vertices, and one writing of the updated vertices. Hence, the I/O cost for *DBP* is given by:

$$\Theta\left(\mathrm{DBP}\left(G\right)\right) = \Theta\left(\frac{\left(\beta+1\right)|V| + |E|}{B} + \beta^2\right) \tag{3.2}$$

In turn, *SPP* performs the following operations over disk: one read of the vertices and one read of the edges grouped by source-partition; next, it shuffles edges by destination-partition in memory, writing the new version $\hat{E}$ on disk; finally, it reads the new edges from disk, calculating the new vertex values and writing them on disk. The I/O cost for *SPP* is:

$$\Theta\left(\mathrm{SPP}\left(G\right)\right) = \Theta\left(\frac{2|V| + |E| + 2\left|\hat{E}\right|}{B} + \beta\right) \tag{3.3}$$

Equations 3.2 and 3.3 define the I/O cost for one processing iteration over the whole graph $G$. However, in order to decide in relation to blocks, we are interested in the costs of Equations 3.2 and 3.3 divided according to the number of blocks $\beta^2$. The result, after the appropriate algebra, reduces to Equations 3.4 and 3.5.

$$\Theta\left(\mathrm{DBP}\left(G^{(p,q)}\right)\right) = \Theta\left(\frac{\vartheta\phi\left(1+1/\beta\right) + \xi\psi}{B}\right) \tag{3.4}$$

$$\Theta\left(\mathrm{SPP}\left(G^{(p,q)}\right)\right) = \Theta\left(\frac{2\vartheta\phi/\beta + 2\xi\phi\psi + \xi\psi}{B}\right) \tag{3.5}$$

in which, $\xi$ is the number of edges in $G^{(p,q)}$, $\vartheta$ is the number of vertices in the interval, and $\phi$ and $\psi$ are, respectively, the number of bytes to represent a vertex and an edge $e$. Once we have the costs per block of DBP and SPP, we can decide between one and the other by simply analyzing the ratio SPP/DBP:

$$\Theta\left(\frac{\mathrm{SPP}}{\mathrm{DBP}}\right) = \Theta\left(\frac{1}{\beta} + \frac{2\xi\phi}{\vartheta}\right) \tag{3.6}$$

This ratio leads to the final decision equation:

$$\mathrm{BlockType}\left(G^{(p,q)}\right) = \begin{cases} \text{sparse}, & \Theta\left(\frac{\mathrm{SPP}}{\mathrm{DBP}}\right) < 1 \\ \text{dense}, & \text{otherwise} \end{cases} \tag{3.7}$$

**Algorithm 3** *MAlgorithm*: Algorithm Interface for coding in M-Flash

```
initialize (Vertex v);
gather (Vertex u, Vertex v, EdgeData data);
process (Accum v_1, Accum v_2, Accum v_out);
apply (Vertex v);
```

**Algorithm 4** PageRank in M-Flash

```
degree(v) = out degree for Vertex v;
initialize (Vertex v):
    v.value = 0;
gather (Vertex u, Vertex v, EdgeData data):
    v.value += u.value/ degree(u);
process (Accum v_1, Accum v_2, Accum v_out):
    v_out = v_1 + v_2;
apply (Vertex v);
    v.value = 0.15 + 0.85 * v.value;
```

We apply Equation 3.6 to select the best option according to Equation 3.7. With this scheme, BBP is able to select the best processing scheme for each block of a given graph. In Section 3.5, we demonstrate that this procedure yields a performance superior than the current state-of-the-art frameworks.

### 3.4.3 Programming Model in M-Flash

M-Flash's computational model, which we named *MAlgorithm* (short for *Matrix Algorithm Interface*) is shown in Algorithm 3. Since *MAlgorithm* is a vertex-centric model, it stores computation results in the destination vertices, allowing for a vast set of iterative graph computations, such as PageRank, Random Walk with Restarts (RWR), Weakly Connected Components (WCC), and diameter estimation.

The *MAlgorithm* interface has four operations: **initialize**, **gather**, **process**, and **apply**. The *initialize* operation loads the initial value of each destination vertex; the *gather* operation collects data from neighboring vertices; the *process* operation processes the data gathered from the neighbors of a given vertex – the desired processing is defined here; finally, the *apply* operation stores the new computed values of the destination vertices to the hard disk, making them available for the next iteration.

*initialize* and *apply* operations are not mandatory and *process* operation is used only in multithreading executions.

To demonstrate the flexibility of *MAlgorithm*, we show in Algorithm 4 the pseudo code of how the PageRank algorithm (using power iteration) can be implemented. The input to PageRank is made of two vectors, one storing node degrees, and another one for storing intermediate PageRank values, initialized to $1/|V|$. The algorithm's output is a third nodes vector that stores

the final computed PageRank values. For each iteration, M-Flash executes the *MAlgorithm* operations on the output vector as follows:

- ***initialize***: the vertices' values are set to 0;
- ***gather***: accumulates the intermediate PageRank values of all in-neighbors $u$ of vertex $v$;
- ***process***: sums up intermediate PageRank values – M-Flash supports multiple threads, so the *process* operation combines the vertex status for threads running concurrently;
- ***apply***: calculates the vertices' new PageRank values (damping factor = 0.85, as recommended by Brian and Sergei [69]).

The input for the next iteration is the output from the current one. The algorithm runs until the PageRank values converge; it may also stop after executing one certain number of iterations defined by the user.

Many other graph algorithms can be decomposed into or take advantage of the same four operations and implemented in similar ways, including Weakly Connected Component (see Algorithm 5), Sparse Matrix Vector Multiplication SpMV (Algorithm 6), eigensolver (Algorithm 7), diameter estimation, and random walk with restart [65].

**Algorithm 5** Weak Connected Component in M-Flash

```
initialize (Vertex v):
    v.value = v.id
gather (Vertex u, Vertex v, EdgeData data):
    v.value = min(v.value, u.value)
process (Accum v_1, Accum v_2, Accum v_out):
    v_out = min(v_1 , v_2)
```

**Algorithm 6** SpMV for weighted graphs in M-Flash

```
initialize (Vertex v):
    v.value = 0
gather (Vertex u, Vertex v, EdgeData data):
    v.value += u.value * data
process (Accum v_1, Accum v_2, Accum v_out):
    v_out = v_1 + v_2
```

### 3.4.4 System Design & Implementation

This section details the implementation of M-Flash. It starts processing the input graph stored in standard file formats, then, it transforms the graph to one flat array format in which each edge has a constant size. At the same time of graph preprocessing, M-Flash divides the edges in $\beta$ *source-partitions* and it counts the number of edges by block. An edge $e = (v_{source}, v_{destination}, data)$

**Algorithm 7** Lanczos Selective Orthogonalization

**Require:** Graph $G(V,E)$
**Require:** dense random vector $\hat{b}$ with size $|V|$
**Require:** maximum number of steps $m$
**Require:** error threshold $\varepsilon$
**Ensure:** Top $k$ eigenvalues $\lambda[1..k]$, eigenvectors $Y^{n\times k}$
  **// M-Flash provides functions for vector**
  **// operations using secondary memory.**
  $\hat{\beta}_0 \leftarrow 0$, $v_0 \leftarrow 0$, $v_1 \leftarrow \hat{b}/\|\hat{b}\|$
  **for** $i = 1$ to $m$ **do**
    $v \leftarrow Gv_i$ **// SpMV using algorithms 8 and 6**
    $\hat{\alpha}_i \leftarrow v_i^T v$
    $v \leftarrow v - \hat{\beta}_{i-1}v_{i-1} - \hat{\alpha}_i v_i$
    $\hat{\beta}_i \leftarrow \|\hat{v}\|$ //Orthogonalization using two previous basis vectors
    $T_i \leftarrow$ (build tri-diagonal matrix from $\hat{\alpha}$ and $\hat{\beta}$)
    $QDQ^T \leftarrow EIG(T_i)$ Eigen decomposition
    **for** $j = 1$ to $i$ **do**
      **if** $\hat{\beta}|Q[i,j]| \leq \sqrt{\varepsilon}\|T_i\|$ **then**
        $r \leftarrow$ selectively orthogonalization using all previous basis vectors $v_1 \ldots v_i$ and $Q[:,j]$
        $v \leftarrow v - (r^T v)r$
      **end if**
    **end for**
    **if** $v$ was selectively orthogonalized) **then**
      $\hat{\beta}_i \leftarrow \|v\|$ //Recompute normalization constant $\hat{\beta}_i$
    **end if**
    **if** $\hat{\beta}_i = 0$ **then**
      break loop
    **end if**
    $v_{i+1} \leftarrow v/\hat{\beta}_i$
  **end for**
  $T \leftarrow$ (build tri-diagonal matrix from $\hat{\alpha}$ and $\hat{\beta}$)
  $QDQ^T \leftarrow EIG(T)$ Eigen decomposition of $T$
  $\lambda[1..k] \leftarrow$ top k diagonal elements of D // $k$ eigenvalues
  $Y \leftarrow V_m Q_k$ // Compute eigenvectors. $Q_k$ is the columns of $Q$ corresponding to $\lambda$

belongs to block $G^{(p,q)}$ when $v_{source} \in I^{(p)}$ and $v_{destination} \in I^{(q)}$. Blocks are classified in sparse or dense using Equation 3.7. Note that M-Flash does **not** sort edges by source or destination, it simply splits edges up to $\beta^2$ blocks, $\beta^2 \ll |V|$. After all edges are preprocessed, whenever a *source-partition* contains dense blocks, M-Flash splits this partition between one sparse partition and dense blocks. The sparse partition contains all edges for the sparse blocks in the *source-partition*. The I/O cost for preprocessing is $\frac{4|E|}{B}$. Algorithm 8 shows the pseudo-code of M-Flash. The aforementioned preprocessing refers to Step 4 of the algorithm. Sparse partitions are processed using *SPP* and dense blocks are processed using *DBP*. Algorithm 8 shows the

overlap between models DBP and SPP, minimizing I/O cost and thus increasing performance.

**Algorithm 8** Main Algorithm of M-Flash

**Require:** Graph $G(V,E)$
**Require:** user-defined *MAlgorithm* program
**Require:** vertex attributes $\gamma$
**Require:** memory size $M$
**Require:** number of iterations *iter*
**Ensure:** vector $v$ with vertex results

```
1: set φ from γ attributes.
2: set β using equation 3.1
3: set ϑ = |V| /β,
4: execute graph preprocessing and partitioning
5: for i = 1 to iter do
6:     Make processing for sparse partitions using SPP
7:     for q = 1 to β do
8:         load vertex values of destination interval I^(q)
9:         initialize I^(q) of v using MAlgorithm.initialize
10:        if exist sparse partition associated to I^(q) then
11:            for each edge
12:                invoke MAlgorithm.gather storing
13:                calculations on vector v
14:        end if
15:        if q is odd then
16:            partition-order = {1 to β}
17:        else
18:            partition-order = {β to 1}
19:        end if
20:        for p = {partition-order} do
21:            if G^(p,q) is dense then
22:                load vertex values of interval I^(p)
23:                for each edge in G^(p,q)
24:                    invoke MAlgorithm.gather storing on v
25:            end if
26:        end for
27:        invoke MAlgorithm.process for I^(q) of v
28:        invoke MAlgorithm.apply for I^(q) of v
29:        store interval I^(q) of vector v
30:    end for
31: end for
```

## 3.5 Evaluation

**Overview:** We compare M-Flash with multiple state-of-the-art approaches: GraphChi, Turbo-Graph, X-Stream, and MMap. For a fair comparison, we use experimental setups recommended by the authors of the other approaches in the best way that we can. We first describe the datasets

Table 4 – Real graph datasets used in our experiments.

| Graph | Nodes | Edges | Size |
|---|---|---|---|
| **LiveJournal** | 4,847,571 | 68,993,773 | Small |
| **Twitter** | 41,652,230 | 1,468,365,182 | Medium |
| **YahooWeb** | 1,413,511,391 | 6,636,600,779 | Large |

used in our evaluation (Subsection 3.5.1) and our experimental setup (Subsection 3.5.2). Then, we compare the runtimes of all approaches for two well-known, essential graph algorithms (Subsections 3.5.3 and 3.5.4) that are available in the competing works. To demonstrate how M-Flash generalizes to more algorithms, we implemented the Lanczos algorithm (with *selective orthogonalization*) as an example, one of the most computationally efficient approach to compute eigenvalues and eigenvectors [70, 65] (Subsection 3.5.5) — to the best of our knowledge, M-Flash provides the **first design and implementation** that can handle graphs with more than one billion nodes when the vertex data cannot fully fit in RAM (e.g., YahooWeb graph). Next, in Subsection 3.5.6, we show that M-Flash continues to run at high speed even when the machine has little RAM (including extreme cases, like when using solely 4GB), in contrast to other methods that slow down in such circumstance. Finally, through an analysis of I/O operations, we show that M-Flash performs far fewer read and write operations than other approaches, which empirically validates the efficiency of our block partitioning model (Subsection 3.5.7).

### 3.5.1 Graph Datasets

We use three real graphs of different scales: a LiveJournal graph [71] with 69 million edges (small), a Twitter graph [72] with 1.47 billion edges (medium), and the YahooWeb graph [7] with 6.6 billion edges (large). Table 4 reports their numbers of nodes and edges.

### 3.5.2 Experimental Setup

All experiments were run on a standard desktop computer with an Intel i7-4770K quad-core CPU (3.50 GHz), 16 GB RAM and 1 TB Samsung 850 Evo SSD disk. Note that M-Flash does *not* require an SSD to run (neither do GraphChi and X-Stream), while TurboGraph does; thus, we used an SSD to make sure all methods can perform at their best. Table 5 shows preprocessing time for each graph using 8GB of RAM.

GraphChi, X-Stream and M-Flash were run on Linux Ubuntu 14.04 (x64). TurboGraph was run on Windows (x64) since it only supports Windows [48]. MMap was written in Java, thus we were able to run it on both Linux and Windows; we ran it on Windows, following MMap's authors setup [67]. All the reported runtimes were given by the average time of three **cold** runs, that is, with all caches and buffers purged between runs to avoid any potential advantage gained due to caching or buffering effects.

Table 5 – Preprocessing time (seconds)

| | Live Journal | Twitter | YahooWeb |
|---|---|---|---|
| **GraphChi** | 23 | 511 | 2781 |
| **X-Stream** | 219 | 5082 | 26200 |
| **TurboGraph** | 18 | 582 | 4694 |
| **MMap** | 17 | 372 | 636 |
| **M-Flash** | 10 | 206 | 1265 |

The libraries are configured as follows:

- GraphChi: C++ version, downloaded from their GitHub repository in February, 2015. Buffer sizes configured to those recommended by their authors[1];
- X-Stream: C++ v0.9. Buffer size desirably configured close to available RAM;
- TurboGraph: V0.1 Enterprise Edition. Buffer size desirably configured to 75% of available RAM, the limit supported by TuboGraph, as observed by [67]);
- MMap: Java version (64-Bit) with default parameters.
- M-Flash: C++ version. `https://github.com/M-Flash`.

We ran all the methods at their best configurations since we wanted to truly verify performance at the most competitive circumstances. As we show in the following sections, M-Flash exceeded the competing works both empirically and theoretically. At the end of the experiments, it became clear that the design of M-Flash considering the density of blocks of the graph granted the algorithm improved performance.

### 3.5.3 PageRank

Figure 12 shows how the PageRank runtime of all the methods compares.

**LiveJournal** (small graph; Fig. 12a): Since the whole graph and all node vectors fully fit in RAM, all approaches finish in seconds. Still, M-Flash was the fastest, up to 3.3X of GraphChi, and 2.4X of X-Stream and TurboGraph.

**Twitter** (medium graph; Fig. 12b): The edges of this graph do not fit in RAM (it requires 11.3GB) but its node vectors do. M-Flash had a similar performance, but for a few seconds, if compared to TurboGraph and MMap for two reasons: (a) the Twitter graph is less challenging as it has a homogeneous density, with all its blocks being sparse; (b) their implementations were highly optimized as they do *not* provide a generic programming model, saving on function calls.

[1] When evaluating how GraphChi performs with 8 GB RAM (Section 3.5.6), we doubled GraphChi's recommended buffer size for 8GB to shrink runtimes.

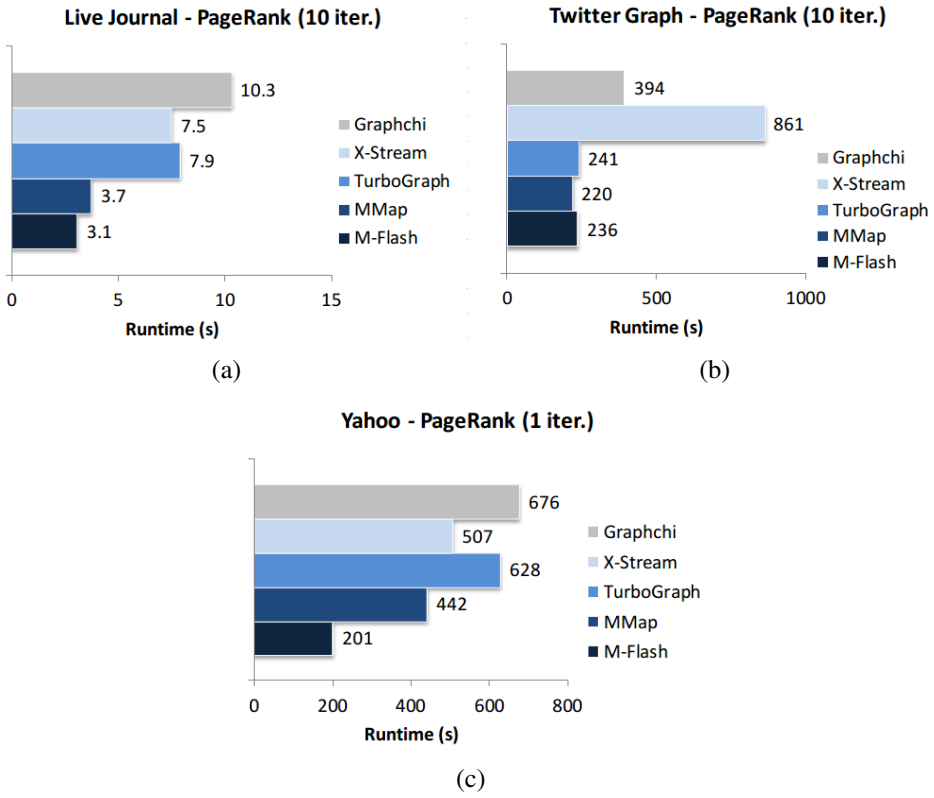



Figure 12 – Runtime of PageRank for LiveJournal, Twitter and YahooWeb graphs with 8GB of RAM. M-Flash is 3X faster than GraphChi and TurboGraph. (a & b): for smaller graphs, such as Twitter, M-Flash is as fast as some existing approaches (e.g., MMap) and significantly faster than other (e.g., 4X of X-Stream). (c): M-Flash is significantly faster than all state-of-the-art approaches for YahooWeb: 3X of GraphChi and TurboGraph, 2.5X of X-Stream, 2.2X of MMap.

In comparison with GraphChi and X-Stream, the related works that offer generic programming models, M-Flash was faster, at 1.7X to 3.6X speed.

**YahooWeb** (large graph; Fig. 12c): For this billion-node graph, neither its edges nor its node vectors fit in RAM; this challenging situation is where M-Flash significantly outperforms the other methods. Figure 12(c) confirms this claim, showing that M-Flash is faster, at a speed that is 2.2X to 3.3X that of the other approaches.

### 3.5.4 Weakly Connected Component

When there is enough memory to store all the vertex data, the *Union Find* algorithm [73] is the best option to find all the *Weakly Connected Components (WCC)* in one single iteration. Otherwise, with memory limitations, an iterative algorithm produces identical solutions, as listed in Appendix A, Algorithm 5. Hence, in this round of experiments, we use Algorithm *Union Find* to solve WCC for the small and medium graphs, whose vertices fit in memory; and we use Algorithm 5 to solve WCC for the YahooWeb graph.

Figures 13(a) and 13(b) show the runtimes for the LiveJournal and Twitter graphs with 8GB RAM; all the approaches use Union Find, except X-Stream. This is because of the way that

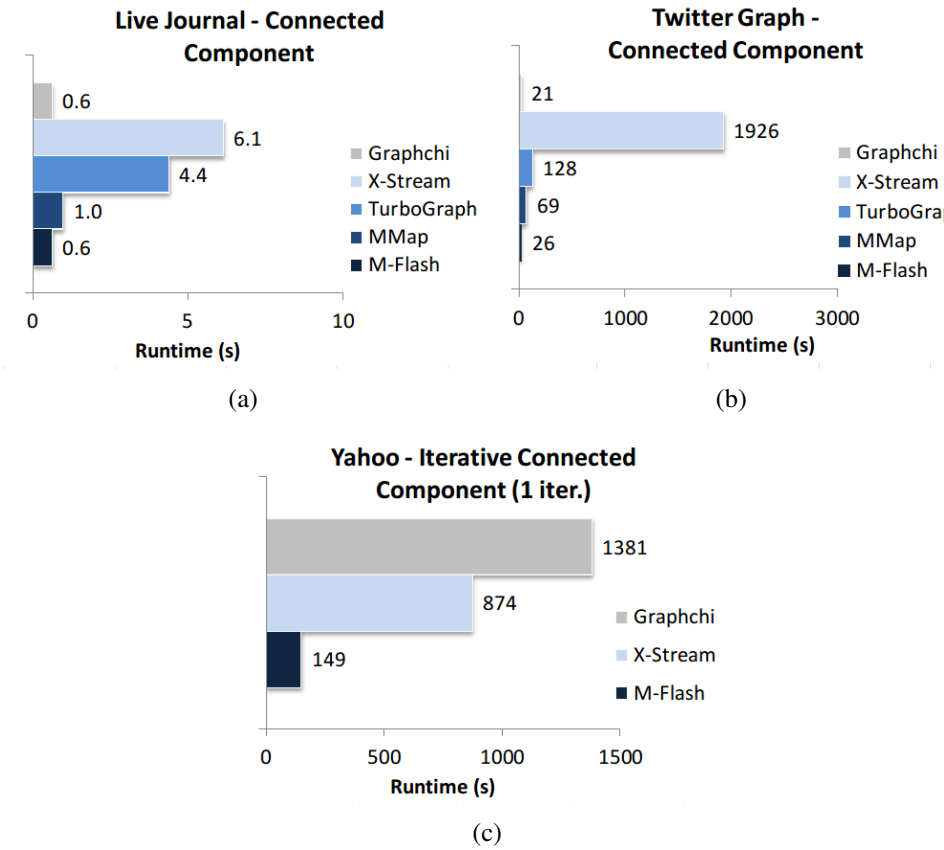



Figure 13 – Runtimes of the Weakly Connected Component problem for LiveJournal, Twitter, and YahooWeb graphs with 8GB of RAM. (a & b): for the small (LiveJournal) and medium (Twitter) graphs, M-Flash is faster than, or as fast as, all the other approaches. (c) M-Flash is pronouncedly faster than all the state-of-the-art approaches for the large graph (YahooWeb): 9.2X of GraphChi and 5.8X of X-Stream.

X-Stream is implemented, which handles only iterative algorithms.

In the WCC problem, M-Flash is again the fastest method in respect to the entire experiment: for the LiveJournal graph, M-Flash ties with GraphChi, it is 9X faster than X-Stream, 7X than TurboGraph, and 1.5X than MMap. For the Twitter graph, M-Flash's speed is only a few seconds behind GraphChi, 74X faster than X-Stream, 5X than TurboGraph, and 2.6X than MMap.

The results for the YahooWeb graph are shown in Figure 13(c), one can see that M-Flash was significantly faster than GraphChi, and X-Stream. Similar to the PageRank results, M-Flash is significantly faster: 9.2X faster than GraphChi and 5.8X than X-Stream.

### 3.5.5 Spectral Analysis using The Lanczos Algorithm

Eigenvalues and eigenvectors are at the heart of numerous algorithms, such as singular value decomposition (SVD) [74], spectral clustering [75], triangle counting [76], and tensor decomposition [77]. Hence, due to its importance, we demonstrate M-Flash over the *Lanczos algorithm*, a state-of-the-art method for eigen computation. We implemented it using method **Selective**

**Orthogonalization** (*LSO*). This implementation demonstrates how M-Flash's design can be easily extended to support spectral analysis of billion-scale graphs. To the best of our knowledge, M-Flash provides the **first design and implementation** that can handle Lanczos for graphs with more than one billion nodes when the vertex data cannot fully fit in RAM. M-Flash provides functions for basic vector operations using secondary memory. Therefore, for the YahooWeb graph, we are not able to compare it with the other competing frameworks using only 8GB of memory, as in the case of GraphChi.

To compute the top 20 eigenvectors and eigenvalues of the YahooWeb graph, one iteration of *LSO* over M-Flash takes 737s when using 8 GB of RAM. For a comparative panorama, to the best of our knowledge, the closest comparable result of this computation comes from the HEigen system [78], at 150s for one iteration; note however that, it was for a much smaller graph with 282 million edges (23X fewer edges), using a **70-machine** Hadoop cluster, while our experiment with M-Flash used a single, commodity desktop computer and a much larger graph.

### 3.5.6 Effect of Memory Size

As the amount of available RAM strongly affects the computation speed in our context, we study here the effect of memory size. Figure 14 summarizes how all approaches perform under 4GB, 8GB, and 16GB of RAM, when running one iteration of PageRank over the YahooWeb graph. M-Flash continues to run at the highest speed even when the machine has very little RAM, 4 GB in this case. Other methods tend to slow down. In special, MMap does not perform well due to *thrashing*, a situation when the machine spends a lot of time on mapping disk-resident data to RAM or un-mapping data from RAM, slowing down the overall computation. For 8 GB and 16 GB, respectively, M-Flash outperforms all the competitors for the most challenging graph, the YahooWeb. Notice that all the methods, but for M-Flash and X-Stream, are strongly influenced by restrictions in memory size; according to our analyses, this is due to the higher number of data transfers needed by the other methods when not all the data fit in the memory. Despite that X-Stream worked well for any memory setting, it still has worse performance if compared to M-Flash because it demands three full disk scans in every case – actually, the innovations of M-Flash, as presented in Section 3.4, come to overcome such problems, which we diagnosed with a series of experiments. In Section 3.5.7, we further elaborate on this allegation.

### 3.5.7 Input/Output (I/O) Operations Analysis

Input/Output (I/O) operations are commonly used as objective measurements for evaluating frameworks based on secondary memory [45].

Figure 15 shows how M-Flash compares with GraphChi and X-Stream in terms of their read and write operations, and the total amount of data read from or written to disk[2] When

[2] We did not compare with TurboGraph because it runs only on Windows, which does not provide readily available tools for measuring I/O speed.

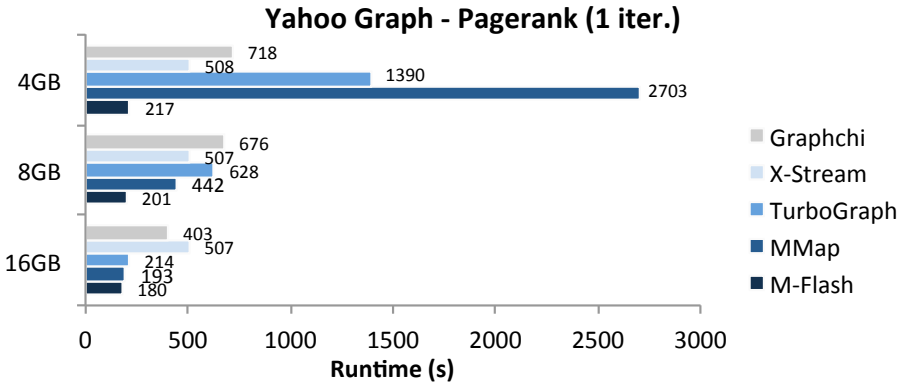



Figure 14 – Runtime comparison for PageRank (1 iteration) over the YahooWeb graph. M-Flash is significantly faster than all the state-of-the-art for three different memory settings, 4 GB, 8 GB, and 16 GB.

running one iteration of PageRank on the 6.6 billion edge YahooWeb graph, M-Flash performs significantly fewer reads (77GB) and fewer writes (17GB) than other approaches. Note that M-Flash achieves and sustains high-speed reading from disk (the "plateau" at the top-right), while other methods do not. For example, GraphChi generally writes data slowly across the whole computation iteration, and X-Stream shows periodic and spiky reads and writes.

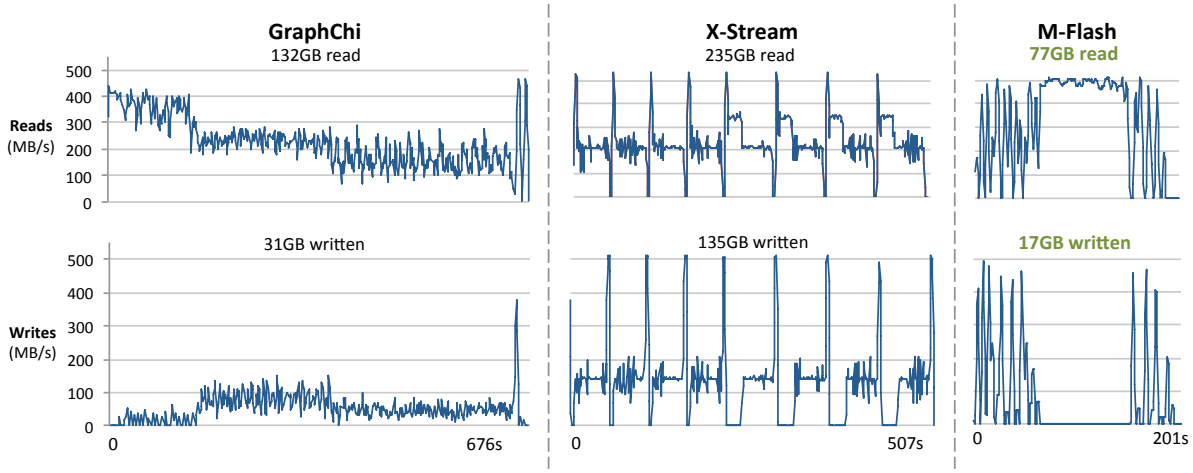



Figure 15 – I/O operations for 1 iteration of PageRank over the YahooWeb graph. M-Flash performs significantly fewer reads and writes (in green) than other approaches. M-Flash achieves and sustains high-speed reading from disk (the "plateau" in top-right), while other methods do not.

### 3.5.8 Theoretical (I/O) Analysis

In the following, we show the theoretical scalability of M-Flash when we reduce the available memory (RAM) at the same time that we demonstrate why the performance of M-Flash improves when we combine DBP and SPP into BBP, instead of using DBP or SSP alone. Here, we use a measure that we named *t-cost*; 1 unit of t-cost corresponds to three operations, one reading of the vertices, one writing of the vertices, and one reading of the edges. In terms of computational

complexity, t-cost is defined as follows:

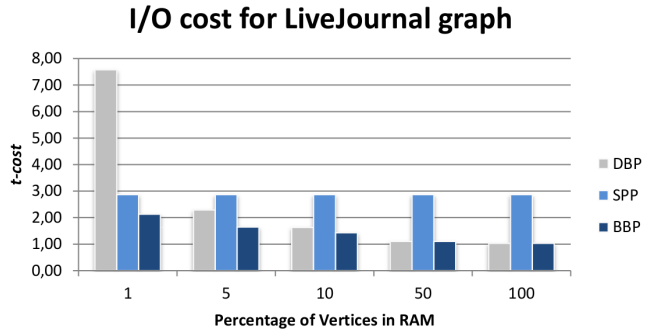
$$\text{t-cost}(G(E,V)) = 2|V| + |E| \tag{3.8}$$

Notice that this cost considers that reading and writing the vertices have the same cost; this is because the evaluation is given in terms of computational complexity. For more details, please refer to the work of McSherry **et al.** [79], who draws the basis of this kind of analysis.

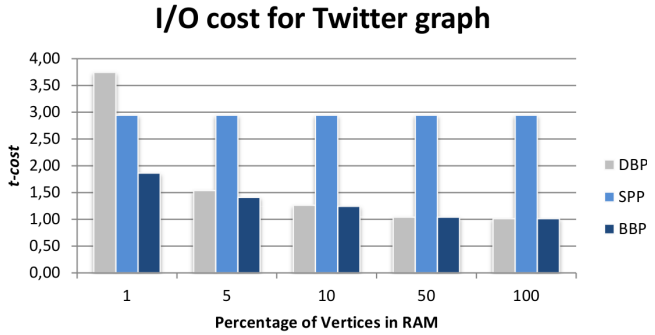
We use measure t-cost to analyze the theoretical scalability for processing schemes *DBP* only, *SPP* only, and *BBP* (the combination of *DBP* and *SPP*). We perform these analyses by means of MathLab simulations that were validated empirically. We considered the characteristics of the three datasets used so far, LiveJournal, Twitter, and YahooWeb. For each case, we calculated the t-cost (y axis) as a function of the available memory (x axis), which, as we have seen, is the main constraint for graph processing frameworks.

Figure 16 shows that, for all the graphs, DBP-only processing is the least efficient when memory is reduced; however, when we combine DBP (for dense region processing) and SPP (for sparse region processing) into BBP, we benefit from the best of both worlds. The result corresponds to the best performance, as seen in the charts. Figure 17 shows the same simulated analysis – t-cost (y axis) in function of the available memory (x axis), but now with an extra variable: the density of hypothetical graphs, which is assumed to be uniform in each analysis. Each plot, from (a) to (d) considers a different density in terms of average vertex degree, respectively, 3, 5, 10, and 30. In each plot, there are two curves, one corresponding to DBP-only, and one for SSP-only; and, in dark blue, we depict the behavior of M-Flash according to the combination BBP. Notice that as the amount of memory increases, so does the performance of DBP (in light graph), which takes less and less time to process the whole graph (decreasing curve). SPP, in turn, has a steady performance, as it is not affected by the amount of memory (light blue line). In dark blue, one can see the performance of BBP; that is, which kind of processing will be chosen by Equation 3.7 at each circumstance. For sparse graphs, Figures 17(a) and 17(b), SSP answers for the greater amount of processing; while the opposite is observed in denser graphs, Figures 17(c) and 17(d), when DBP defines almost the entire dark blue line of the plot.

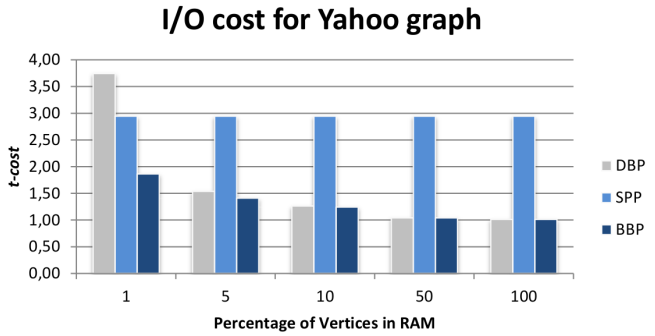
These results show that the graph processing must take into account the density of the graph at each moment (block) so to choose the best strategy. It also explains why M-Flash improves the state-of-the-art. It is **important** to note that no former algorithm considered the fact that most graphs present varying density of edges (dense regions with many more edges than other regions that are sparse). Ignoring this fact leads to decreased performance in the form of higher number of data transfers between memory and disk, as we empirically verified in the former sections.




(a)




(b)




(c)

Figure 16 – I/O cost using *DBP*, *SPP*, and *BBP* for LiveJournal, Twitter and YahooWeb Graphs using different memory sizes. *BBP* model always performs fewer I/O operations on disk for all memory configurations.

## 3.6 Conclusions

We proposed M-Flash, a **single-machine**, **billion-scale** graph computation framework that uses a block partition model to maximize disk access speed. M-Flash uses an innovative design that takes into account the variable density of edges observed in the different blocks of a graph. Its design uses Dense Block Processing (DBP) when the block is dense, and Streaming Partition Processing (SPP) when the block is sparse; for taking advantage of both worlds, it uses the combination of DBP and SPP according scheme Bimodal Block Processing (BBP), which is able to analytically determine whether a block is dense or sparse and trigger the appropriate processing. To date, M-Flash is the first framework that considers a bimodal approach for I/O minimization.

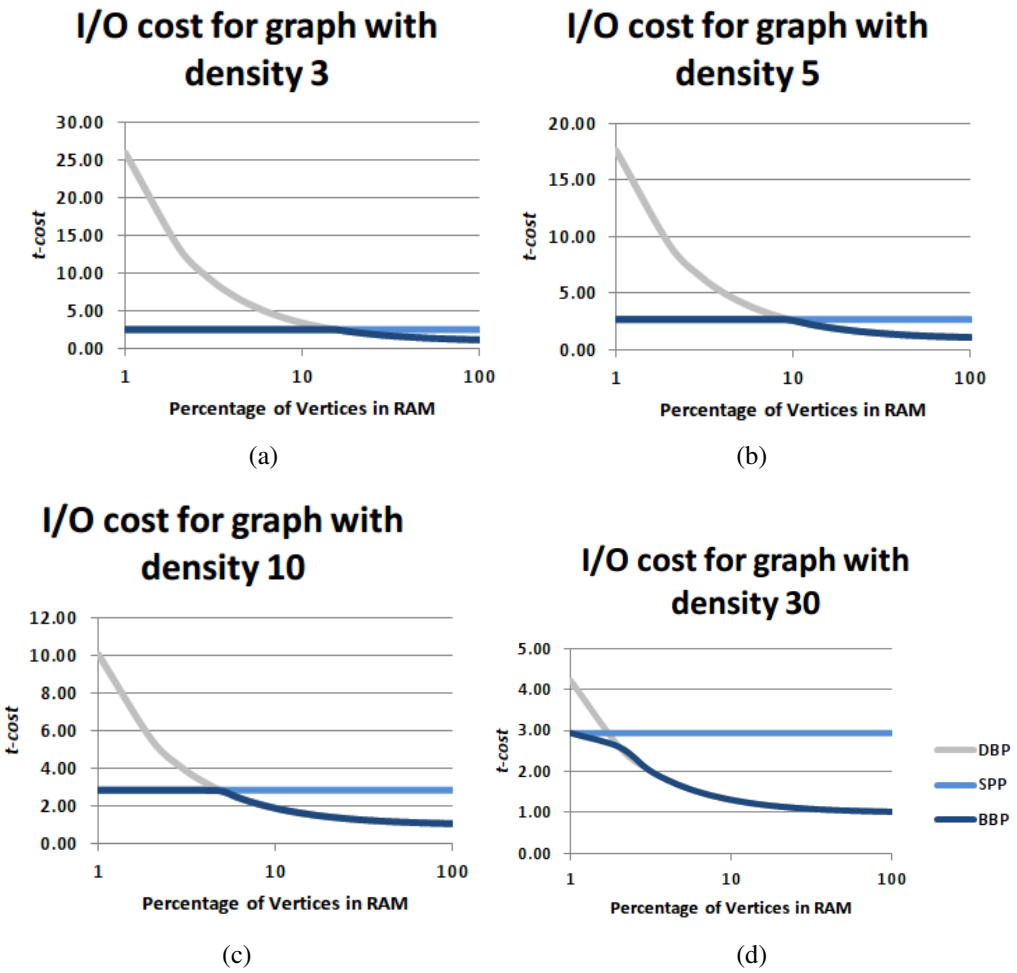



Figure 17 – I/O cost using DBP, SPP, and BBP for a graph with densities $k = \{3, 5, 10, 30\}$. Graph density is the average vertex degree, $|E| \approx k|V|$. *DBP* increases considerably I/O cost when RAM is reduced, *SPP* has a constant I/O cost and *BBP* chooses the best configuration considering the graph density and available RAM.

M-Flash was designed so that it has possible integrated a wide range of popular graph algorithms according to its Matrix Algorithm Interface model, including the *first* single-machine billion-scale eigensolver. We conducted extensive experiments using large real graphs. M-Flash consistently and significantly outperformed all state-of-the-art approaches, including GraphChi, X-Stream, TurboGraph and MMap. M-Flash runs at high speed for graphs of all sizes, including the 6.6 billion edge YahooWeb graph, even when the size of memory is very limited (e.g., 6.4X as fast as X-Stream, when using 4GB of RAM).

## 3.7 Final considerations

In this chapter we presented our framework M-Flash for fast processing of billion-scale graphs. We performed extensive experiments on real graphs with up to 6.6 billion edges, demonstrating M-Flash's consistent and significant speed-up over state-of-the-art approaches. Over our new methodology, we introduced three contributions: (1) a novel framework solution; (2) a scalable

approach that works faster that other frameworks when resources are severely constrained (e.g., 6.4X faster than X-Stream, when using 4GB of RAM; and (3) a flexible and simple programming interface that allows implementation of popular and essential algorithms, including the *first* single-machine billion-scale eigensolver.

CHAPTER

# 4

# FURTHER RESULTS

## 4.1 Initial considerations

On sections 2 and 3, we described the main contributions of this project, proposing algorithms and a framework for an efficient processing and visualization of large graphs. Additionally, we got complementary results over a specific domain, the DBLP co-authoring and co-citation network. We proposed a multimodal analysis that ensembles statistical, algorithmic, algebraic and topological techniques of data analysis, and we discovered non-evident facts about the DBLP repository. Hence, this chapter summarizes contributions of two published works in collaboration: "Supervised-learning link recommendation in the DBLP co-authoring network" [1] and "Multimodal graph-based analysis over the dblp repository: critical discoveries and hypotheses" [2].

## 4.2 Multimodal analysis of DBLP

Scientific collaboration is a type of complex network formed from research relations between academic and industrial pairs with common motivations to understand and explain natural and synthetic phenomenons. The behavior and trends of these networks are relevant not only for authors and editors, but the funding agencies and the society that demands knowledge about how scientists behave concerning their collective production.

There are plenty of studies on scientific collaboration networks. Newman [80] showed that scientific communities have "small-world" phenomena and co-authoring networks are highly clustered, so two scientists are much more likely to have collaborated when they have common partners; Leydesdorff [81] evaluated the interdisciplinarity found in journals using measures like degree, betweenness, and closeness; although inconclusive, the author brings light to the problem. Osiek *et al.*[82] try to answer whether conferences influence on research collaboration, and authors lead the conclusion that conferences do not significantly promote collaboration among

participants (Only 4.61% of collaborations between authors satisfied their supposition). J. Huang *et al.* [83] performed a three-level analysis (network level, community level, and individual level) with a fragment of the Computer Science CiteSeer Digital Library [1]; they introduce a stochastic Poisson model to predict future collaboration behavior. These previous works characterize the properties of nodes alone or, at least, the global properties of the structure by means of single metrics. We analyze the DBLP data by drawing statistical distribution for several of its properties, and by drawing metrics that consider the time dimension.

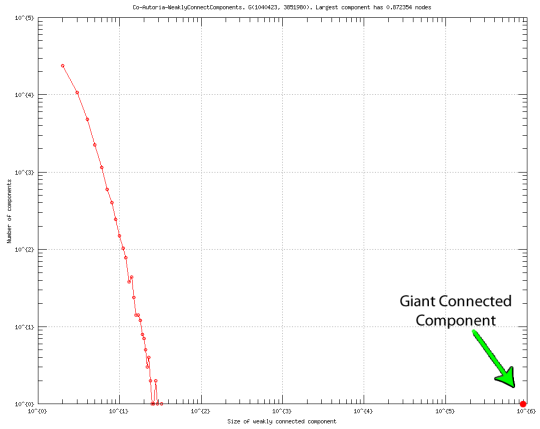
| **Relation** | **Description** |
|---|---|
| *Co-authorship* | *Authors who published papers together.* |
| *Co-participation* | *Authors who had papers in the same conference.* |
| *Co-publication* | *Authors who had papers in the same journal.* |
| *Co-edition* | *Authors who appeared as editors of the same event or journal.* |

Table 6 – Relations extracted from DBLP and used in our analysis. [1]

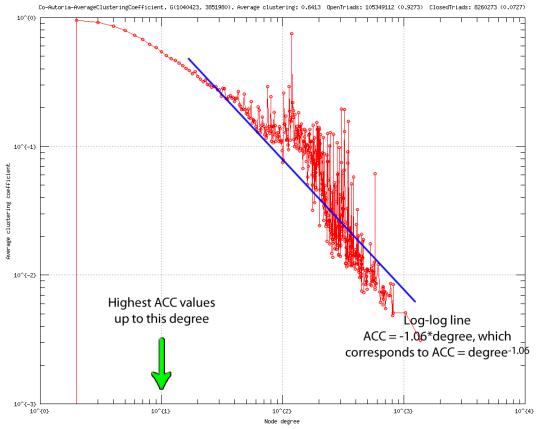
We used of a number of social network metrics and techniques to inspect the characteristics of DBLP in complementary fashion. We extracted four datasets that reflect the distinct relations between researchers (see Table 6). We verified interesting facts about Weakly-connected component distribution (WCC), figure 18a depicts the case for co-authorship. We observed only 13% of the authors are part of small components with up to 30 relations and 87% of authors ( $10^6$ nodes) define a giant community, sharing scientific expertise. A common property of co-authoring graphs is significant values of the average clustering coefficient (ACC). We verified this property (see Figure 18b) and we observed that high values (close to 1) only happens for nodes with degree up to around 10. Also, we noted the power law $ACC \propto degree^{-1.06}$; this ACC behavior occurs because authors tend to collaborate with co-authors of their co-authors, as was suggested by Newman [80]. In addition, older authors (advisors) are less likely to be part of one well-defined and highly interconnected community, and they tend to be connected with multiple subgraphs with low densification (degree distribution) in time.

From the degree-distribution plot (see Figure 18c), we inspected that DBLP's degree distribution obeys a power law with exponent $\gamma \approx -1.36$. However, one might wonder why this phenomenon occurs in co-authoring networks where edge creation is more expensive than in other environments like a web network. We presume two facts to explain this tendency. First, master and Ph.D. titles became regular courses in the last decades with well-defined time schedules and expected production; therefore, a demand for "where to publish", rather than "what to publish", was created. Second, science-funding agencies have demanded a lot of divulgation in order to keep their financing. Hence, a straight consequence of this fact is the increasing number of authors per paper; in some cases, it doesn't imply an intellectual guidance and labor, but to co-financing and personal exchange as a means to increase one's production.

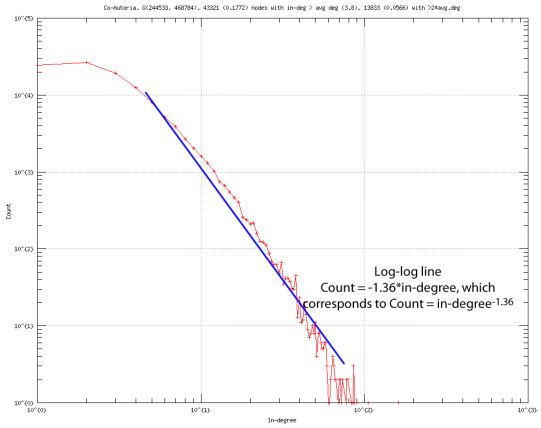
[1] http://citeseer.ist.psu.edu



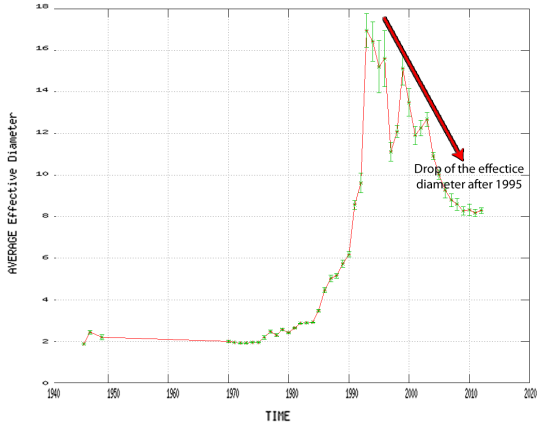

(a) Co-authorship weakly-connected components (WCC) distribution.




(b) Co-authorship Average Clustering Coefficient (ACC) against node degree.




(c) In-degree distribution of the co-authorship activity in DBLP – the higher the in-degree (the co-authorship), the smaller the number of authors according to power law $Count \propto indegree^{-1.36}$.




(d) Co-edition effective diameter evolution.

Figure 18 – This results were published in our paper [1].

From the diameter evolution of the co-edition network (see Figure 18d), we observed that the effective diameter has started to shrink around the year 1995. A possible explanation is that the committees of editors tend to have the same members that alternate each year between a small set of possible committees; thence, the distance between editors tends to decrease along the time. Additionally, the editing of publications is an activity that demand higher experience and expertise, characteristics of a limited group of researchers.

Statistical (degree, and weakly-connected component distribution), topological (average clustering coefficient, and effective diameter evolution) techniques helped us to understand essential properties of DBLP from a static and dynamic (time-evolve) analysis; however, we were interested in how predictable is the DBLP using machine learning classifiers. We used the link recommendation techniques to predict new co-authoring relations using topological metrics of authors: number of common neighbors, Jaccard's coefficient, Preferential attachment,

Adamic-Adar coefficient, Resource allocation index, and Local path. We considered different profile of authors with at least $d$ existing co-authors. This extra parameter allows a comparison about the predictability of the less ($d \geq 1$) and the more ($d \geq 8$) active authors. We compared the accuracy for supervised machine learning classifiers [84] J48, Naïve Bayes, Multilayer Perceptron, Bagging, and Random Forest. Please check our work [1] for details. Figure 19 shows that authors with more collaborations are more predictable than authors with little ones. Less active authors possibly correspond to researchers with few partners or casual researchers that abandon the academy after getting their degree.

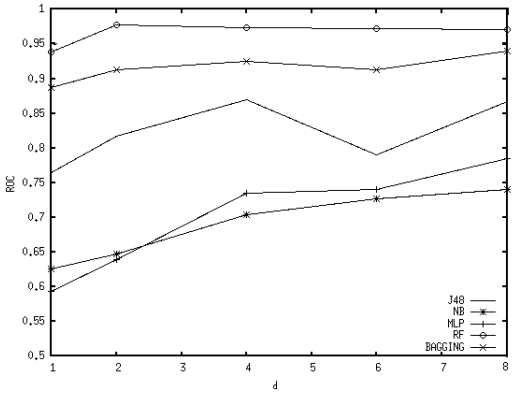



Figure 19 – AUC visualization of the data generated for a co-authoring snapshot of DBLP between [1995, 2007]. We used collaborations between [1995, 2005] to predict new ones between [2006, 2007].

In our last analysis of the DBLP, we considered an algebraic approach and we proposed the metric *Sao Paulo's importance* as expressed by equation 4.1. Our metric has two variables, quantifying how active (*accomplishment*) and constant in time (*silence*) the author is; researchers with high *accomplishment* are more productive in their career (number of publications) and researchers with high silence (time since the last publication) are less active at the moment of the measurement. Figure 20 shows the behavior of *SP' importance* and the counting (3D histogram) of authors using *silence* and *accomplishment* variables on the DBLP repository. In this figure, most of the authors have low *silence* ($\leq 5$ years) and low accomplishment ($\leq 5$ years); these authors are possibly students that are still doing their graduation course, or they have recently finished it. This finding reveals how Computer Science is dependent on casual researchers, and also how competitive it is, since just a few authors are able to migrate to the more important region of the plot.

$$SP'Importance = \frac{1}{\sqrt{silence+1}} * log(Accomplishment) \tag{4.1}$$

## 4.3 Final considerations

We proposed a multimodal analytical approach assembling statistical (degree, and weakly-connected component distribution), topological (average clustering coefficient, and effective

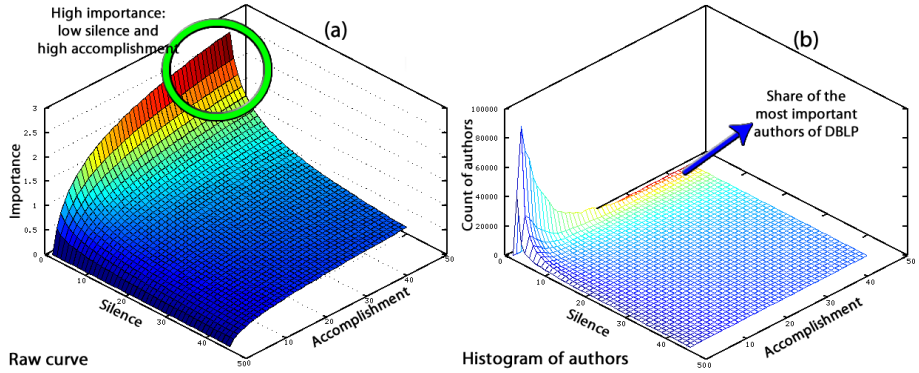



Figure 20 – Plot of metric $SP'Importance$. (a) Raw curve of Equation 4.1. (b) Counting (3D histogram) of authors in relation to the possible values of metric $Importance$. [2]

diameter evolution), algorithmic (link prediction/machine learning), and algebraic techniques to reveal non-evident features of network-like data, including networks of co-authoring, recommendation, computer routing, social interaction, protein interaction, to name a few. We demonstrated our process over the DBLP repository of Computer Science publications introducing an innovative course of action based on techniques that, although apart, can be used in complementary fashion.

CHAPTER

# 5

# CONCLUSIONS

Processing and visualization of billion-scale graphs are big challenges commonly tackled with distributed frameworks. However, the deployment and management of computational clusters can be complex, demanding technical and financial resources that can be prohibitive in a variety of scenarios. Therefore, it is desirable to have techniques for processing and visualization of large-scale graphs with resource optimization in a single computational node. In this work, consequently, we developed an ample set of single-node techniques to support processing and visualization in analytical tasks over large (million and billion scale) graphs.

In chapter 2 we worked on the hypothesis that **relations between recurrent and simple patterns characterize graph domains providing interesting insights through exploratory visualization**. Accordingly, Section 2.5 showed evidences of the applicability of StructMatrix as a highly scalable methodology for visual inspection of graph structures. Additionally, our results revealed macro patterns and interest findings on real-life graphs for several domains.

In chapter 3 we presented a methodology to verify the hypothesis that **a framework focused on minimizing I/O communication is able to boost the processing speed of planetary-scale graphs that do not fit in RAM**, as detailed in the experiments presented in Section 3.5. Our results demonstrated that, by using the innovative bimodal block processing model (BBP), our framework was able to outperform all the competitors for single-node processing; our results soundly demonstrated the flexibility, scalability, and robustness of our solution.

Finally, in chapter 4 we showed our further results about an analytical approach that ensembles statistical, topological, algorithmic, and algebraic techniques to reveal non-evident features in co-authoring networks. Although the experiments occurred over a co-authoring network, our methodology is extensible to other problems, such as product recommendation, social interaction, computer routing, and protein interaction.

In this work we developed and experimented with visual and analytical techniques for efficient processing of large-scale graphs. As discussed, we obtained interesting results; still,

future works are still to be sought:

- In our first contribution, the implementation of StructMatrix considers graphs only in-memory. Nonetheless, we expect to combine techniques StructMatrix and M-Flash so to extend visualization capabilities to graphs in secondary memory;
- In our second contribution, it would be interesting to add support for asynchronous and incremental graph computation in a way similar to GraphChi; with these further development, we expect to reach even higher performance.

# BIBLIOGRAPHY

[1] G. P. Gimenes, H. Gualdron, T. R. Raddo, and J. F. Rodrigues, "Supervised-learning link recommendation in the dblp co-authoring network," in *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2014 IEEE International Conference on*, pp. 563–568, IEEE, 2014. Cited 6 times on pages 14, 19, 65, 66, 67, and 68.

[2] G. P. Gimenes, H. Gualdron, and J. F. Rodrigues, "Multimodal graph-based analysis over the dblp repository: critical discoveries and hypotheses," in *Proceedings of the ACM Symposium on Applied Computing*, pp. 1–7 (to appear), ACM Press, 2015. Cited 3 times on pages 15, 65, and 69.

[3] I. Twitter, "#numbers." `https://blog.twitter.com/2011/numbers`, 2011. Maio, 2015. Cited on page 23.

[4] V. Gudivada, R. Baeza-Yates, and V. Raghavan, "Big data: Promises and problems," *Computer*, vol. 48, pp. 20–23, Mar 2015. Cited on page 23.

[5] GE and Accenture, "Industrial internet insights report for 2015." `https://www.accenture.com/Accenture-Landscape-Industries.pdf`. novembro, 2014. Cited on page 23.

[6] B. Krishnamurthy, P. Gill, and M. Arlitt, "A few chirps about twitter," in *Proceedings of the First Workshop on Online Social Networks*, WOSN '08, (New York, NY, USA), pp. 19–24, ACM, 2008. Cited on page 23.

[7] Yahoo!Labs., "Yahoo altavista web page hyperlink connectivity graph, 2002.," 2002. Accessed: 2014-12-01. Cited 3 times on pages 23, 44, and 55.

[8] C. Liu, F. Guo, and C. Faloutsos, "Bbm: bayesian browsing model from petabyte-scale data," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 537–546, ACM, 2009. Cited on page 23.

[9] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: A system for large-scale graph processing," in *Proc. of the 2010 ACM SIGMOD*, SIGMOD '10, (New York, NY, USA), pp. 135–146, ACM, 2010. Cited 4 times on pages 24, 25, 43, and 45.

[10] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein, "Distributed graphlab: A framework for machine learning and data mining in the cloud," *Proc. VLDB Endow.*, vol. 5, pp. 716–727, Apr. 2012. Cited 2 times on pages 24 and 25.

[11] A. Roy, I. Mihailovic, and W. Zwaenepoel, "X-stream: Edge-centric graph processing using streaming partitions," in *Proc. of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP '13, (New York, NY, USA), pp. 472–488, ACM, 2013. Cited 5 times on pages 24, 25, 44, 45, and 49.

[12] W. de Nooy, A. Mrvar, and V. Batagelj, *Exploratory social network analysis with Pajek*, vol. 27. Cambridge University Press, 2005. Cited on page 24.

[13] J. Abello, F. van Ham, and N. Krishnan, "ASK-GraphView: A Large Scale Graph Visualization System," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 12, no. 5, pp. 669–676, 2006. Cited on page 24.

[14] N. Elmqvist, T.-N. Do, H. Goodell, N. Henry, and J. Fekete, "Zame: Interactive large-scale graph visualization," in *PacificVIS*, pp. 215–222, 2008. Cited 2 times on pages 24 and 28.

[15] J. F. Rodrigues Jr., H. Tong, A. J. M. Traina, C. Faloutsos, and J. Leskovec, "GMine: a system for scalable, interactive graph visualization and mining," in *Proceedings of the 32nd international conference on Very large data bases*, VLDB '06, pp. 1195–1198, VLDB Endowment, 2006. Cited on page 24.

[16] A. Perer, I. Guy, E. Uziel, I. Ronen, and M. Jacovi, "Visual social network analytics for relationship discovery in the enterprise.," in *IEEE VAST*, pp. 71–79, IEEE, 2011. Cited on page 24.

[17] A. Perer, I. Guy, E. Uziel, I. Ronen, and M. Jacovi, "The Longitudinal Use of SaNDVis: Visual Social Network Analytics in the Enterprise," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 19, no. 7, pp. 1095–1108, 2013. Cited on page 24.

[18] D. H. Chau, L. Akoglu, J. Vreeken, H. Tong, and C. Faloutsos, "Interactively and Visually Exploring Tours of Marked Nodes in Large Graphs," in *Advances in Social Networks Analysis and Mining (ASONAM), 2012 IEEE/ACM International Conference on*, pp. 696–698, 2012. Cited on page 24.

[19] E. M. Kornaropoulos and I. G. Tollis, "DAGView: an approach for visualizing large graphs," in *Proceedings of the 20th international conference on Graph Drawing*, GD'12, (Berlin, Heidelberg), pp. 499–510, Springer-Verlag, 2013. Cited on page 24.

[20] A. Perer and B. Shneiderman, "Integrating Statistics and Visualization: Case Studies of Gaining Clarity During Exploratory Data Analysis," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '08, (New York, NY, USA), pp. 265–274, ACM, 2008. Cited on page 24.

[21] C. Tominski, J. Abello, and H. Schumann, "CGV-An interactive graph visualization system," *Computers & Graphics*, vol. 33, no. 6, pp. 660–678, 2009. Cited on page 24.

[22] J. Heer and A. Perer, "Orion: A system for modeling, transformation and visualization of multidimensional heterogeneous networks," in *Visual Analytics Science and Technology (VAST), 2011 IEEE Conference on*, pp. 51–60, IEEE, 2011. Cited on page 24.

[23] Y. Perez, R. Sosic, A. Banerjee, R. Puttagunta, M. Raison, P. Shah, and J. Leskovec, "Ringo: Interactive graph analytics on big-memory machines," *CoRR*, vol. abs/1503.07881, 2015. Cited on page 24.

[24] R. Pienta, J. Abello, M. Kahng, and D. H. Chau, "Scalable graph exploration and visualization: Sensemaking challenges and opportunities," in *Big Data and Smart Computing (BigComp), 2015 International Conference on*, pp. 271–278, IEEE, 2015. Cited on page 24.

[25] B. Shneiderman, "The eyes have it: a task by data type taxonomy for information visualizations," in *Visual Languages, 1996. Proceedings., IEEE Symposium on*, pp. 336–343, Sep 1996. Cited on page 24.

[26] D. A. Keim, "Visual exploration of large data sets," *Communications of the ACM*, vol. 44, no. 8, pp. 38–44, 2001. Cited on page 24.

[27] D. A. Keim, "Information visualization and visual data mining," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 8, no. 1, pp. 1–8, 2002. Cited on page 24.

[28] K. Börner, C. Chen, and K. W. Boyack, "Visualizing knowledge domains," *Annual review of information science and technology*, vol. 37, no. 1, pp. 179–255, 2003. Cited on page 24.

[29] J. Heer and D. Boyd, "Vizster: visualizing online social networks," in *Information Visualization, 2005. INFOVIS 2005. IEEE Symposium on*, pp. 32–39, Oct 2005. Cited on page 24.

[30] U. Kang, J.-Y. Lee, D. Koutra, and C. Faloutsos, "Net-ray: Visualizing and mining billion-scale graphs," in *Advances in Knowledge Discovery and Data Mining*, pp. 348–361, Springer, 2014. Cited 2 times on pages 24 and 29.

[31] F. van Ham and A. Perer, "#x0201C;Search, Show Context, Expand on Demand #x0201D;: Supporting Large Graph Exploration with Degree-of-Interest," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 15, no. 6, pp. 953–960, 2009. Cited on page 24.

[32] J. Teevan, C. Alvarado, M. S. Ackerman, and D. R. Karger, "The perfect search engine is not enough: a study of orienteering behavior in directed search," in *Proceedings of the*

*SIGCHI conference on Human factors in computing systems*, pp. 415–422, ACM, 2004. Cited on page 24.

[33] D. H. Chau, A. Kittur, J. I. Hong, and C. Faloutsos, "Apolo: Making Sense of Large Network Data by Combining Rich User Interaction and Machine Learning," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, (New York, NY, USA), pp. 167–176, ACM, 2011. Cited on page 24.

[34] L. Akoglu, D. H. Chau, C. Faloutsos, N. Tatti, H. Tong, J. Vreeken, and L. A. J. V. H. Tong, "Mining connection pathways for marked nodes in large graphs.," in *SDM*, pp. 37–45, SIAM, 2013. Cited on page 24.

[35] L. Akoglu, D. H. Chau, U. Kang, D. Koutra, and C. Faloutsos, "OPAvion: mining and visualization in large graphs," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, (New York, NY, USA), pp. 717–720, ACM, 2012. Cited on page 24.

[36] J. Abello, S. Hadlak, H. Schumann, and H. Schulz, "A Modular Degree-of-Interest Specification for the Visual Analysis of Large Dynamic Networks," 2013. Cited on page 24.

[37] T. von Landesberger, A. Kuijper, T. Schreck, J. Kohlhammer, J. J. van Wijk, J.-D. Fekete, and D. W. Fellner, "Visual Analysis of Large Graphs: State-of-the-Art and Future Research Challenges," *Computer Graphics Forum*, vol. 30, no. 6, pp. 1719–1749, 2011. Cited on page 24.

[38] U. Kang, H. Tong, J. Sun, C.-Y. Lin, and C. Faloutsos, "Gbase: a scalable and general graph management system," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 1091–1099, ACM, 2011. Cited on page 25.

[39] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, pp. 107–113, Jan. 2008. Cited on page 25.

[40] T. White, *Hadoop: The definitive guide*. " O'Reilly Media, Inc.", 2012. Cited on page 25.

[41] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "Powergraph: Distributed graph-parallel computation on natural graphs," in *Proc. of the 10th USENIX Conf. on Operating Systems Design and Implementation*, OSDI'12, (Berkeley, CA, USA), pp. 17–30, USENIX Association, 2012. Cited 3 times on pages 25, 43, and 45.

[42] A. Ching, "Scaling apache giraph to a trillion edges," *Facebook Engineering blog*, 2013. Cited on page 25.

[43] B. Shao, H. Wang, and Y. Li, "Trinity: A distributed graph engine on a memory cloud," in *Proceedings of the 2013 international conference on Management of data*, pp. 505–516, ACM, 2013. Cited on page 25.

[44] R. S. Xin, J. E. Gonzalez, M. J. Franklin, and I. Stoica, "Graphx: A resilient distributed graph system on spark," in *First International Workshop on Graph Data Management Experiences and Systems*, GRADES '13, (New York, NY, USA), pp. 2:1–2:6, ACM, 2013. Cited on page 25.

[45] A. Kyrola, G. Blelloch, and C. Guestrin, "Graphchi: Large-scale graph computation on just a pc," in *Proc. of the 10th USENIX Conf. on Operating Systems Design and Implementation*, OSDI'12, (Berkeley, CA, USA), pp. 31–46, USENIX Association, 2012. Cited 5 times on pages 25, 43, 44, 45, and 59.

[46] A. Khan and S. Elnikety, "Systems for big-graphs," *Proc. VLDB Endow.*, vol. 7, pp. 1709–1710, Aug. 2014. Cited 2 times on pages 25 and 45.

[47] D. M. Da Zheng, R. Burns, J. Vogelstein, C. E. Priebe, and A. S. Szalay, "Flashgraph: processing billion-node graphs on an array of commodity ssds," Cited on page 25.

[48] W.-S. Han, S. Lee, K. Park, J.-H. Lee, M.-S. Kim, J. Kim, and H. Yu, "Turbograph: A fast parallel graph engine handling billion-scale graphs in a single pc," in *Proc. of the 19th ACM SIGKDD*, KDD '13, (New York, NY, USA), pp. 77–85, ACM, 2013. Cited 5 times on pages 25, 43, 44, 45, and 55.

[49] E. Yoneki and A. R. 0002, "Scale-up graph processing: a storage-centric view.," in *GRADES* (P. A. Boncz and T. N. 0001, eds.), p. 8, CWI/ACM, 2013. Cited 2 times on pages 25 and 45.

[50] D. Holten, "Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data," *IEEE TVCG*, vol. 12, no. 5, pp. 741–748, 2006. Cited on page 28.

[51] D. H. Chau, A. Kittur, J. I. Hong, and C. Faloutsos, "Apolo: making sense of large network data by combining rich user interaction and machine learning," in *SIGCHI Conference on Human Factors in Computing Systems*, pp. 167–176, ACM, 2011. Cited on page 28.

[52] U. Kang, C. E. Tsourakakis, and C. Faloutsos, "Pegasus: A peta-scale graph mining system implementation and observations," in *Data Mining, 2009. ICDM'09. Ninth IEEE International Conference on*, pp. 229–238, IEEE, 2009. Cited on page 28.

[53] L. Akoglu, M. McGlohon, and C. Faloutsos, "Oddball: Spotting anomalies in weighted graphs," in *Advances in Knowledge Discovery and Data Mining*, pp. 410–421, Springer, 2010. Cited on page 28.

[54] E. Bertini and G. Santucci, "By chance is not enough: preserving relative density through nonuniform sampling," in *Information Visualisation*, pp. 622–629, IEEE, 2004. Cited on page 28.

[55] M. Ghoniem, J. Fekete, and P. Castagliola, "A comparison of the readability of graphs using node-link and matrix-based representations," in *IEEE InfoVis*, pp. 17–24, 2004. Cited on page 28.

[56] J. Abello and F. van Ham, "Matrix zoom: A visual interface to semi-external graphs," in *IEEE InfoVis*, pp. 183–190, 2004. Cited on page 28.

[57] N. Henry and J.-D. Fekete, "Matlink: Enhanced matrix visualization for analyzing social networks," in *Int Conference on Human-computer Interaction*, pp. 288–302, Springer-Verlag, 2007. Cited on page 29.

[58] N. Henry, J. Fekete, and M. J. McGuffin, "Nodetrix: a hybrid visualization of social networks," *IEEE TVCG*, vol. 13, no. 6, pp. 1302–1309, 2007. Cited on page 29.

[59] D. Chakrabarti, Y. Zhan, D. Blandford, C. Faloutsos, and G. Blelloch, "Netmine: New mining tools for large graphs," in *SIAM-DM Workshop on Link Analysis*, 2004. Cited on page 29.

[60] B. A. Prakash, A. Sridharan, M. Seshadri, S. Machiraju, and C. Faloutsos, "Eigenspokes: Surprising patterns and scalable community chipping in large graphs," in *Advances in Knowledge Discovery and Data Mining*, pp. 435–448, Springer, 2010. Cited on page 29.

[61] D. Lasalle and G. Karypis, "Multi-threaded graph partitioning," in *IEEE Int Symp on Parallel and Distributed Processing*, pp. 225–236, 2013. Cited on page 29.

[62] D. Koutra, U. Kang, J. Vreeken, and C. Faloutsos, "Vog: Summarizing and understanding large graphs," in *Proceedings of the SIAM International Conference on Data Mining (SDM), Philadelphia, PA*, SIAM, 2014. Cited on page 29.

[63] U. Kang and C. Faloutsos, "Beyond 'caveman communities': Hubs and spokes for graph compression and mining," in *ICDM*, pp. 300–309, 2011. Cited on page 29.

[64] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney, "Statistical properties of community structure in large social and information networks," in *WWW*, pp. 695–704, 2008. Cited on page 30.

[65] U. Kang, C. E. Tsourakakis, and C. Faloutsos, "Pegasus: A peta-scale graph mining system implementation and observations," in *Proc. of the 2009 Ninth IEEE Int. Conf. on Data Mining*, ICDM '09, (Washington, DC, USA), pp. 229–238, IEEE Computer Society, 2009. Cited 4 times on pages 43, 45, 52, and 55.

[66] K. Munagala and A. Ranade, "I/o-complexity of graph algorithms," in *Proc. of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '99, (Philadelphia, PA, USA), pp. 687–694, Society for Industrial and Applied Mathematics, 1999. Cited on page 44.

[67] Z. Lin, M. Kahng, K. M. Sabrin, D. H. Chau, H. Lee, and U. Kang, "Mmap: Fast billion-scale graph computation on a pc via memory mapping," in *BigData*, 2014. Cited 4 times on pages 44, 45, 55, and 56.

[68] A. Aggarwal and S. Vitter, Jeffrey, "The input/output complexity of sorting and related problems," *Commun. ACM*, vol. 31, pp. 1116–1127, Sept. 1988. Cited on page 49.

[69] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," *Computer networks and ISDN systems*, vol. 30, no. 1, pp. 107–117, 1998. Cited on page 52.

[70] B. N. Parlett and D. S. Scott, "The lanczos algorithm with selective orthogonalization," *Mathematics of computation*, vol. 33, no. 145, pp. 217–238, 1979. Cited on page 55.

[71] L. Backstrom, D. Huttenlocher, J. Kleinberg, and X. Lan, "Group formation in large social networks: Membership, growth, and evolution," in *Proc. of the 12th ACM SIGKDD*, KDD '06, (New York, NY, USA), pp. 44–54, ACM, 2006. Cited on page 55.

[72] H. Kwak, C. Lee, H. Park, and S. Moon, "What is twitter, a social network or a news media?," in *Proc. of the 19th Int. Conf. on World Wide Web*, WWW '10, (New York, NY, USA), pp. 591–600, ACM, 2010. Cited on page 55.

[73] R. E. Tarjan and J. van Leeuwen, "Worst-case analysis of set union algorithms," *J. ACM*, vol. 31, pp. 245–281, Mar. 1984. Cited on page 57.

[74] M. W. Berry, "Large-scale sparse singular value computations," *International Journal of Supercomputer Applications*, vol. 6, no. 1, pp. 13–49, 1992. Cited on page 58.

[75] U. Von Luxburg, "A tutorial on spectral clustering," *Statistics and computing*, vol. 17, no. 4, pp. 395–416, 2007. Cited on page 58.

[76] C. E. Tsourakakis, "Fast counting of triangles in large real networks without counting: Algorithms and laws," in *Proc. of the 2008 Eighth IEEE Int. Conf. on Data Mining*, ICDM '08, (Washington, DC, USA), pp. 608–617, IEEE Computer Society, 2008. Cited on page 58.

[77] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM Review*, vol. 51, no. 3, pp. 455–500, 2009. Cited on page 58.

[78] U. Kang, B. Meeder, E. E. Papalexakis, and C. Faloutsos, "Heigen: Spectral analysis for billion-scale graphs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 2, pp. 350–362, 2014. Cited on page 59.

[79] F. McSherry, M. Isard, and D. G. Murray, "Scalability! but at what cost," Cited on page 61.

[80] M. E. Newman, "The structure of scientific collaboration networks," *Proceedings of the National Academy of Sciences*, vol. 98, no. 2, pp. 404–409, 2001. Cited 2 times on pages 65 and 66.

[81] L. Leydesdorff, "Betweenness centrality as an indicator of the interdisciplinarity of scientific journals," *Information Science and Technology*, vol. 58, pp. 1303–1309, 2006. Cited on page 65.

[82] B. A. Osiek, G. Xexeo, A. S. Vivacqua, and J. M. de Souza, "Does conference participation lead to increased collaboration? a quantitative investigation," *IEEE Computer Supported Cooperative Work in Design*, pp. 642–647, 2009. Cited on page 65.

[83] J. Huang, Z. Zhuang, J. Li, and C. L. Giles, "Collaboration over time: characterizing and modeling network evolution," in *Web search and web data mining*, pp. 107–116, 2008. Cited on page 66.

[84] I. H. Witten, E. Frank, and M. A. Hall, *Data Mining - Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2011. Cited on page 68.