A Science Gateway to Support Research in Spectral Graph Theory[*]

Daniel Oliveira[1], **Carlos Magno Abreu**[1], **Eduardo Ogasawara**[1], **Eduardo Bezerra**[1], **Leonardo de Lima**[2]

[1]Federal Center of Technological Education of Rio de janeiro - CEFET/RJ, Brazil

[2]Paraná Federal University - UFPR, Curitiba, PR, Brazil

{daniel.oliveira, ebezerra, eogasawara}@cefet-rj.br

leonardo.delima@ufpr.br, magno.mabreu@gmail.com

Abstract. *Describing classes of graphs that optimize a function of the eigenvalues subject to some constraints is one of the topics addressed by Spectral Graph Theory (SGT). In this paper, we propose RioGraphX, a science gateway developed on top of Apache Spark, which aims to obtain all graphs that optimize a given mathematical function of the eigenvalues of a graph. Initial experiments involving small graphs have pointed out optimal graphs in a reasonable computational time, and also have shown that leveraging parallel processing is a promising approach to handle larger graphs.*

1. Introduction

Spectral Graph Theory (SGT) is a research area which aims to obtain structural properties of a graph from eigenvalues and eigenvectors of matrices related to its graph. Given a graph $G(V, E)$ with vertex set V of cardinality n and edge set E of cardinality m, a set of matrices can be associated with G. The most used representations for graphs in SGT are the adjacency, the Laplacian, and the signless Laplacian matrices. The largest and the smallest non-zero eigenvalues of those matrices are strongly related to structural properties of the graph. For instance, one can cite the second smallest Laplacian eigenvalue of a graph, called the algebraic connectivity of the graph. This parameter is associated with the connectivity of the graph such that G is connected if and only if the algebraic connectivity of G is positive [Mohar et al., 1991]. For other connections on the eigenvalues of the adjacency, Laplacian and signless Laplacian to invariants of graphs, we refer the reader to [Cvetkovic et al., 2007], [Wilf, 1967], and [Bomze et al., 1999]. The SGT community has increased over the last years after the very first work of Dragos Cvetkovic thesis [Cvetković, 1971]. Since then, applications in many areas have been reported, and in particular in Computer Science [Cvetković and Simić, 2011]. Computational tools aiming to help in either proposing or refuting conjectures and describing families of graphs satisfying some properties have been developed in the last few years. For instance, the AutoGraphiX [Caporossi and Hansen, 2000] is a heuristic tool which has been used to solve many open problems in the literature and has led to the publication of over 20 papers, all of them related to problems of SGT. This fact shows how computer systems that aid theoretical researchers can be useful to propose the right mathematical conjectures or to refute

[*]Os autores agradecem à FAPERJ, à CAPES (código 001) e ao CNPq pelo financiamento do projeto.

them. Other relevant computational tools in SGT are NewGraph [Brankov et al., 2006], MathChem [Vasilyev and Stevanović, 2014], and Graph6Java [Ghebleh et al., 2019]. It is worth mentioning that all these computational tools for SGT run monolithically on desktop computers and none of them are available online. Besides, most of them require some level of coding in a specific programming language. In order to verify a conjecture, one may code routines generating all graphs for a given range of n vertices and look for counterexamples, which is a very time-consuming process. The limitations of these tools can be seen in Table 1. In fact, due to the lack of scalability, evaluating conjectures for graphs with ten or more vertices becomes a computational barrier while using these tools.

In this paper, we propose RioGraphX, a science gateway published as a Web application to aid SGT researchers either in the investigation of counterexamples for an existing conjecture or in describing classes of graphs that optimize a given function under some possible constraints. RioGraphX does an exhaustive search to find graphs in a given range of orders that optimize some user-provided combinatorial function (also known as extremal graphs). A workflow defined in the Apache Spark environment [Zaharia et al., 2016] searches for extremal graphs using a parallel and distributed computational infrastructure, by calculating their invariants and producing a ranked list of graphs that optimize the given function. This ranked list is finally presented to the researcher for further analysis. RioGraphX implements a workflow which integrates well-known packages, such as NetworkX [Hagberg et al., 2008], GraphX [Xin et al., 2013], and GraphFrames [Dave et al., 2016] and is innovative in the sense that it makes use of a scientific workflow approach with distributed and parallel processing. It is worth mentioning that Paralell BGL [Gregor and Lumsdaine, 2005] is another alternative of parallel and distributed graph packages, which we intend to explore in future works. Interesting features of RioGraphX include: (i) no usage of processing resources in the user's machine; (ii) provides an interactive online user interface, where the users can formulate mathematical conjectures and test them without coding computational routines; (iii) users can monitor the execution of their submissions. Also, a single user can submit several jobs simultaneously and all results are displayed in the report folder area; (iv) provides a PDF report containing the k top extremal graphs that optimize a mathematical function given as an input by the user; (v) makes use of parallel and distribution execution (through Spark) to generate all graphs that optimize a given function with some predefined constraints. We conducted initial computational experiments using a conjecture proposed in the literature for graphs with different orders and obtained reasonable speedup results.

2. The RioGraphX Science Gateway

The RioGraphX is a science gateway built on top of Spark environment and makes usage of Python, Nauty package, NetworkX, and GraphFrames. Table 1 presents the RioGraphX functionalities compared to five similar tools. It can be seen that it has the main features of other tools and also presents ease of use, accessibility (via a Web interface) and parallel distributed execution (integrated with Spark). The workflow of activities executed by RioGraphX is summarized in Figure 1. The RioGraphX deals with a huge computational challenge of a potentially large number of graphs to be processed, even for small graphs for each job submission. In Figure 1, we describe the activities of the RioGraphX workflow, and how it uses Spark parallelization to distribute the processes. All invariants of a graph implemented in the system are described in Table 2.

Table 1. Functional characteristics of SGT computational tools

Functionalities	NewGraph	MathChem	AutoGraphiX	Graph6Java	RioGraphX
Manipulation of g6 files	No	Yes	Yes	Yes	Yes
Interactive environment	Yes	No	Yes	No	Yes
Programming skills needed	No	Yes	No	Yes	No
Parallel and distributed execution	No	No	No	No	Yes
Accessible from anywhere	No	No	No	No	Yes
Search for extremal graphs	No	No	Yes	Yes	Yes

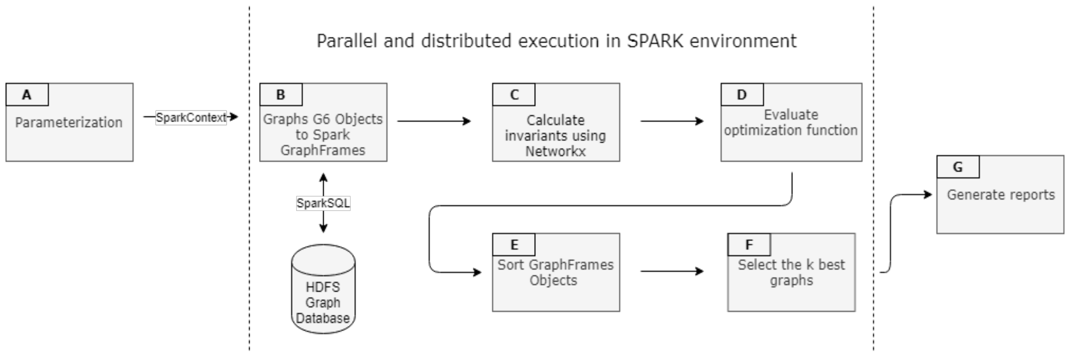

Parallel and distributed execution in SPARK environment
A
Parameterization
SparkContext
B
Graphs G6 Objects to Spark GraphFrames
SparkSQL
HDFS Graph Database
C
Calculate invariants using Networkx
D
Evaluate optimization function
E
Sort GraphFrames Objects
F
Select the k best graphs
G
Generate reports

Figure 1. The workflow implemented in RioGraphX

The front-end of RioGraphX consists of a Java WEB application where users can create a RioGraphX account. Once authenticated, the user has access to the submission form (see Figure 2) where configuration parameters are filled to submit the job. In step **A** of the workflow, the user configures the experiment (job) related to the conjecture to be assessed (see Figure 2). First, the analytical form of a function is typed in Latex format. More specifically, the function has the type $f(x_1, x_2, \ldots, x_t)$, where each x_i, for $i = 1, 2, \ldots, t$ can be one of the following graph invariants of the Table 2. After defining the function f, the optimization type (either maximizing or minimizing f) is defined and also the number k of graphs that should either minimize or maximize f. Table 3 shows the available constraints to be added to the problem. Such constraints decrease the search space of graphs to be selected.

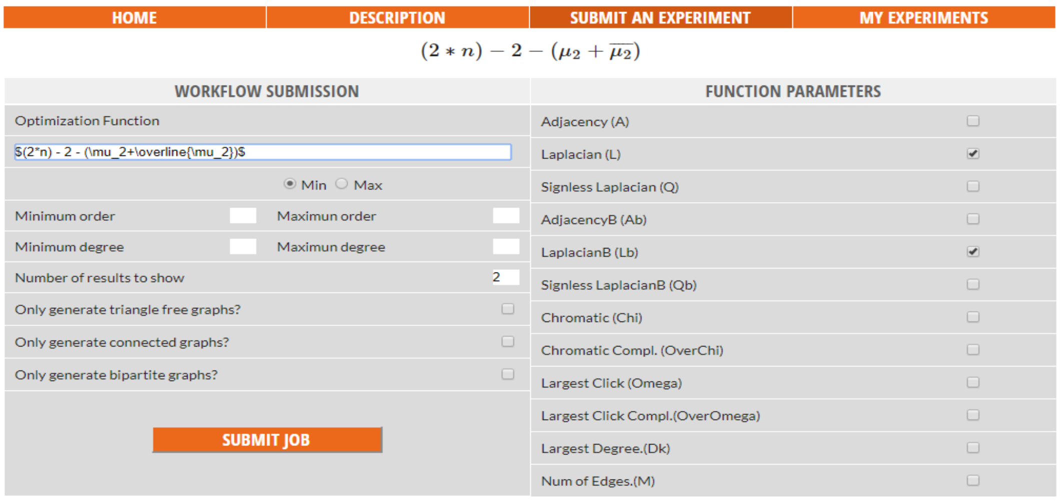
Through an account, the user can monitor the submission status (processing or terminated) of all requests already made. Other screens of the system are not presented here due to lack of space. Once a job submission is placed into the system, RioGraphX

Table 2. List of parameters available for the definition of the optimization function

Graph invariants	Description
$[n_{min}, n_{max}]$	minimum and maximum order of a graph
d_i	i-th largest degree of a graph
λ_i	i-th largest eigenvalue of a graph
μ_i	i-th largest Laplacian eigenvalue of a graph
q_i	i-th largest signless Laplacian eigenvalue of a graph
χ_G	chromatic number of a graph
ω_G	clique number of a graph

Table 3. Available constraints in RioGraphX system

Constraints	Description
Graphs free of triangles	generation of triangle-free graphs only
Connected Graphs	generation of connected graphs only
Bipartite Graphs	generation of bipartite graphs only
k	number of graphs to be included in the report after execution



$(2 * n) - 2 - (\mu_2 + \overline{\mu_2})$

WORKFLOW SUBMISSION		FUNCTION PARAMETERS	
Optimization Function		Adjacency (A)	☐
`$(2*n) - 2 - (\mu_2+\overline{\mu_2})$`		Laplacian (L)	☑
[✓ Min] Max		Signless Laplacian (Q)	☐
Minimum order	Maximun order	AdjacencyB (Ab)	☐
Minimum degree	Maximun degree	LaplacianB (Lb)	☑
Number of results to show	2	Signless LaplacianB (Qb)	☐
Only generate triangle free graphs?	☐	Chromatic (Chi)	☐
Only generate connected graphs?	☐	Chromatic Compl. (OverChi)	☐
Only generate bipartite graphs?	☐	Largest Click (Omega)	☐
		Largest Click Compl.(OverOmega)	☐
		Largest Degree.(Dk)	☐
		Num of Edges.(M)	☐

Figure 2. Dynamic form for job submission

runs a remote procedure to generate the corresponding graphs and, for each one of them, the optimization function is evaluated according to the steps of Figure 1 represented in the boxes between the vertical dashed lines. In step **B**, all simple graphs of order n in the range $n_{min} \leq n \leq n_{max}$ are generated. It is the object creation phase and one of the costliest steps, in which RioGraphX make use of distributed and parallel processing on Spark. In order to save the obtained results, the system maintains a cache of all simple connected graphs of orders ranging from 4 to 10 in the HDFS database. Table 4 presents the number of graphs according to its order. In the worst case, all of those graphs should be stored in this cache. For each graph, the following information is stored: (i) a string in g6 format

Table 4. Number of connected graphs and graphs with orders from 4 to 10

Order	4	5	6	7	8	9	10
All graphs	11	34	156	1,044	12,346	274,668	12,005,168
Connected Graphs	6	21	112	853	11,117	261,080	11,716,571
% of connected graphs	54,5	61,8	71,8	81,7	90,0	95,0	97,6

representing the graph; (ii) the number of vertices; (iii) minimum degree; (iv) maximum degree; (v) an attribute indicating whether the graph is triangle free or not; (vi) an attribute indicating whether the graph is connected or not; (vii) an attribute indicating whether the graph is bipartite or not. This data is loaded into the Spark processing stream through the SparkSQL module [Armbrust et al., 2015]. In step **C**, also in parallel and distributed mode between the Spark nodes, the computation of each invariant of the optimization function is done by using the NetworkX library. In step **D**, the optimization function is calculated and evaluated for each generated graph. Step **E** sorts the set of generated graphs, according to the corresponding optimization function value obtained in the previous step and, in step **F**, the k best graphs are selected. Finally, step **G** produces a summary report in a PDF with information about each graph obtained during the search process, with their bitmap images, which correspond to the parameters informed in step **A**.

3. Experimental Evaluation

The proposed architecture for testing is composed of the main node, responsible for storing the Tomcat service (which provides the WEB interface), a PostgreSQL database, and the Spark Master environment (responsible for administering the worker nodes which perform the processing in a distributed and parallel fashion). All nodes are deployed using dockers with the Alpine Linux distribution as the operating system. The master node has 20GB of RAM, and each Worker has 20GB of RAM and six processing cores. A Python algorithm implementing the workflow until step F was done, and speedup tests were performed in Spark. The speedup tests measured the average time of the system to give the optimal solution for graphs ranging from 5 to 10 vertices. Our example considered the inequality $\mu_2(G) + \mu_2(\overline{G}) \leq 2n - 2$, where $\overline{G}$ is the graph complement of G. In order to test whether this inequality is true, we have used the function $f(\mu_2, \overline{\mu_2}, n) = n - 2 - (\mu_2(G) + \mu_2(\overline{G}))$ and required only connected graphs. After minimization, if RioGraphX returns a graph such that $f(\mu_2, \overline{\mu_2}, n) \leq 0$, we have a counterexample, and so the conjecture will be disproved. On the other hand, if $f(\mu_2, \overline{\mu_2}, n) \geq 0$ for all graphs we have a stronger indication that conjecture might be true. Also, we can take the graphs where $f(\mu_2, \overline{\mu_2}, n) = 0$ and extend the conjecture presenting the extremal graphs. After running RioGraphX for all graphs ranging from 5 to 10 vertices, no counterexamples were found and the obtained extremal graphs motivated the statement of Conjecture 5 in [Grijó et al., 2019]. Table 5 shows the results of the tests with different numbers of workers (nodes) where we can see that the proposed conjecture was confirmed in times considered optimal given the size of the dataset used (see Table 4), a large number of calculations performed and the ordering of the results. We verified that the time gain is linear between the results of one and two nodes. Linearity is lost as nodes were added but execution times remain significant.

Table 5. Speedup tests for graphs with 5 to 10 vertices

Nodes	Average execution time (seconds)	Speedup
1 Node (6 cores)	1817,178 ± 0.316	1,00
2 Nodes (12 cores)	943,937 ± 5,867	1,92
4 Nodes (24 cores)	663,982 ± 41,966	2,73
8 Nodes (48 cores)	472,471 ± 16,541	3,84

4. Conclusion

We proposed RioGraphX, a science gateway to aid SGT researchers in the investigation of accurate and detailed results of graphs and their properties that meet a given function. The corresponding workflow comprises seven well-defined steps that are executed in parallel and distributed inside Spark with the integration of other important tools.

Based on speedup tests, the power of the Spark applied to the proposed workflow already demonstrates that the RioGraphX science gateway is in the path to achieve its goal in becoming an important tool for SGT studies. There are several alternatives for future work. First, we plan to define a more efficient search algorithm, with the possible use of some metaheuristic, in order to find extremal graphs that minimize/maximize the user-provided combinatorial optimization function. We also plan to investigate how to maximize parallel and distributed processing with the GraphFrames libraries and evaluate their performance. Another goal is to evaluate performance and quality in the calculation of invariants in larger graphs (with more than ten vertices). Moreover, we intend to investigate speedup behavior with the use of more than eight nodes to optimize the execution time.

References

Armbrust et al. (2015). Spark sql: Relational data processing in spark. ACM.

Bomze, I. M., Budinich, M., Pardalos, P. M., and Pelillo, M. (1999). The maximum clique problem. In Handbook of combinatorial optimization, pages 1–74. Springer.

Brankov, V., Cvetković, D., Simić, S., and Stevanović, D. (2006). Simultaneous editing and multilabelling of graphs in system newgraph.

Caporossi, G. and Hansen, P. (2000). Variable neighborhood search for extremal graphs: 1 the autographix system. Discrete Mathematics, 212(1-2):29–44.

Cvetkovic, D., Rowlinson, P., and Simic, S. K. (2007). Eigenvalue bounds for the signless laplacian. Publications de l'Institut Mathématique, 81(95):11–27.

Cvetković, D. and Simić, S. (2011). Graph spectra in computer science. Linear Algebra and its Applications, 434(6):1545–1562.

Cvetković, D. M. (1971). Graphs and their spectra. Publikacije Elektrotehničkog fakulteta. Serija Matematika i fizika, (354/356):1–50.

Dave et al. (2016). Graphframes: an integrated api for mixing graph and relational queries. In Proceedings of the Fourth International Workshop on Graph Data Management Experiences and Systems, page 2. ACM.

Ghebleh, M., Kanso, A., and Stevanović, D. (2019). Graph6java: A researcher–friendly java framework for testing conjectures in chemical graph theory. MATCH.

Gregor, D. and Lumsdaine, A. (2005). The parallel bgl: A generic library for distributed graph computations. Parallel Object-Oriented Scientific Computing (POOSC), 2:1–18.

Grijó, R. et al. (2019). Nordhaus–gaddum type inequalities for the two largest laplacian eigenvalues. Discrete Applied Mathematics, 267:176–183.

Hagberg, A., Swart, P., and S Chult, D. (2008). Exploring network structure, dynamics, and function using networkx. Technical report, LANL.

Mohar, B., Alavi, Y., Chartrand, G., and Oellermann, O. (1991). The laplacian spectrum of graphs. Graph theory, combinatorics, and applications, 2(871-898):12.

Vasilyev, A. and Stevanović, D. (2014). Mathchem: a python package for calculating topological indices. MATCH Commun. Math. Comput. Chem, 71:657–680.

Wilf, H. S. (1967). The eigenvalues of a graph and its chromatic number. Journal of the London mathematical Society, 1(1):330–332.

Xin, R. S., Gonzalez, J. E., Franklin, M. J., and Stoica, I. (2013). Graphx: A resilient distributed graph system on spark. ACM.

Zaharia et al. (2016). Apache spark: A unified engine for big data processing. Commun. ACM, 59(11):56–65.