Enhancement on the Labeled Component Unfolding system for parallel implementations

Lucas Francisco Amaral Orosco Pellicer
Polytechnic School of São Paulo (EPUSP)
University of São Paulo (USP)
São Paulo, SP, Brazil
E-mail: lucas.pellicer3394@gmail.com

Filipe Alves Neto Verri, Vitor Venceslau Curtis
Computer Science Division (IEC)
Aeronautics Institute of Technology (ITA)
São José dos Campos, SP, Brazil
E-mail: verri@ita.br, curtis@ita.br

***Abstract*—Machine learning algorithms are in the fastest-growing fields of interest in recent years. In this work, a semi-supervised learning algorithm based on complex networks is adapted to exploit distributed processing of vertices. This is an algorithm based on the propagation of particles generated by labeled vertices to other edges of this complex network. Particles are absorbed at the edges and provide a dominance relationship between the vertices of the network and how the classes make up the problem. However, this algorithm is formulated by some sums that are performed throughout the database, which decreases the speed-up in parallel implementation due to underutilization. Our work focuses on stipulating an estimator of these values of this sum that decreases the need of the reduction. We demonstrate the formulation of the estimator and we show that the modification increases the classification performance of the original algorithm while increasing the parallelization potential of the algorithm.**

***Index Terms*—Semi-supervisioned learning, complex networks, computational efficiency, graph processing.**

I. Introduction

Semi-supervised Learning (SSL) is a machine learning paradigm that combines elements of methods with supervision and unsupervised learning [1].

In real situations, it is common not to have complete knowledge of the desired target variables of our problem [2]. For example, in the context of image processing where you want to identify objects in the image, it is usually unrealistic to have full knowledge of the presence of every object in the image. Another example is to evaluate customer satisfaction where only a small part of them responds to a satisfaction survey [3]. In other situations, it is only possible to know all of our information if a large amount of data is labeled manually [4]. However, this labeling process is, in many situations, tiring and very expensive. In situations like these, the SSL techniques are the most indicated to solve these problems, because they combine the two paradigms of learning and are able to work around this restriction of data [5].

Throughout a few years of study, many SSL techniques have been developed: generative models [3], transductive models techniques [4] and graph-based methods [6], [7]. Of all the approaches already mentioned, graph-based methods have proved to be very interesting for solving many issues [8]. In such techniques, a graph is constructed from the input dataset. To solve the semi-supervised classification problem, the graph-based techniques "propagate" the labels of the few labeled samples to every other vertex in the graph by using an optimization framework [5].

Our work is carried out on a step of the SSL classification algorithm in graphs called Network Unfolding map [2]. As part of the classification process, the algorithm simulates a complex system named Label Component Unfolding (LCU) [2]. This system can retrieve label information of the unlabeled data by analysing the complex network constructed from the input dataset [9]. Here, we call complex network a sparse, undirected, weightless, simple graph with nontrivial connection between vertices [10].

The LCU system models a propagation of labeled particles on the complex network structure. In this way, particles compete for the domination of the edges [2]. This dominance relationship indicates that certain edges are more influenced by some label [2], [11]. This information is further used to classify the unlabeled data of the original partially labeled dataset.

The core of the system computation corresponds to the simulation of particle propagations over the network, imposing computational challenges especially for large graphs, e.g., proper dynamic partitioning of the vertices or edges among the computing units of a cluster in order to reduce the communication latency. In addition, many calculations performed on graphs are costly, usually in the cubic order [5].

During the simulation, at every time step (called epoch), dynamics of the LCU system is performed by realizing several reductions in all edges and vertices of the network. As any reduction of n elements takes a minimum of $\log_2(n)$ steps independently of the computing availability, as shown in Figure 1, it may drastically decrease the speed-up of a large parallel system due to underutilization.

In this work, we propose a modification of the original LCU system that increases the speed performance in parallel implementation by removing a sum reduction. We show that the proposed change increases the classification performance and the exploration behavior of the original technique.

In section II and III, we review the concepts of graph-parallel systems and the formulation of the original LCU system. In section IV, we formulate an estimator to drop a critical sum reduction of the LCU system. Finally, we conclude

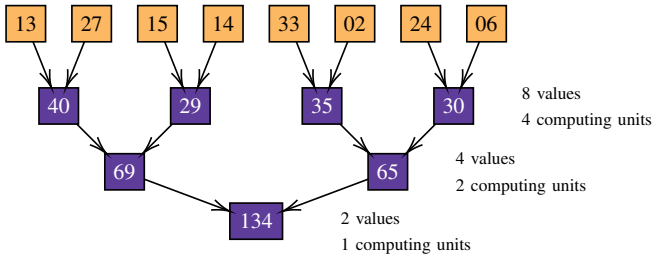

Fig. 1. System utilization on reductions.

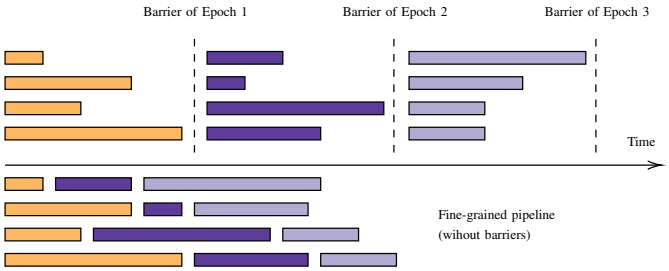

Fig. 2. Dependent tasks without synchronization barrier.

the paper in section VI.

II. Graph-parallel systems review

Modern parallel systems apply many different techniques in order to hide the latency introduced by the dependencies among calculations or tasks. Currently, Spark and Apache Hadoop are the two most famous cluster systems for large data. Hadoop is based on the MapReduce programming model, which implements batch processing with intermediate computing persisted on disks for recovery purpose [12].

Apache Spark exposes a dataflow programming model through a programming interface and it is usually deployed on top of other distributed ecosystems in order to take advantage of their low layers [13]. For example, the computing resources can be managed by the solutions Hadoop YARN, Mesos, Kubernetes and it may access data on distributed file systems as Hadoop Distributed File System (HDFS), Apache Cassandra, and others.

The Resilient Distributed Datasets (RDD) is the distributed memory abstraction that Spark uses to create a Directed Acyclic Graph (DAG) of coarse-grained functional operations, which is used to compose a fine-grained data flow [14]. This approach is better than batch-based model for iterative problems because it does not require barrier synchronizations between iterations (epochs). Instead, just necessary dependencies are included on very small portions of data, called partitions. Figure 2 shows the effect of introducing barrier synchronizations instead of the dependencies among partitions.

Pregel is an another programming model based on dataflow suitable for iterative computing and initially proposed to solve large graph problems. Basically, it implements a parallel message-passing interface among the vertices of a graph [15].

Although Pregel is a proprietary system from Google, many other graph-parallel systems have been emerged in recent years based on it. GraphX, a Spark API which implements the Pregel framework, can be used to expose an easy programming interface for graph problems while taking advantage of all the previous mentioned features of the Spark ecosystem [16]. Thus, in this article, we implement the LCU using the GraphX API of the Spark ecosystem to check the performance of our solution.

Both the complexity of the problem and dependencies are considered to be great impediments to the implementation of the LCU by parallel processing. Thus, in this article, we propose a new ϕ estimator which simplifies the system equations and eliminates some reductions of the LCU. In this way, we can decrease the complexity of the operations involved at each iteration of the algorithm and remove possible bottleneck in the parallelization of the LCU. We will compare the original algorithm with our proposal to indicate that we get a formulation that leaves the algorithm more efficient, but without a performance cost in the classification.

III. Model Review

LCU is a dynamical system that runs over a complex network whose vertices are partially labeled. In the system, labeled particles propagate in the network at random and compete against rival particles (that is, particles with different labels) for edges. As a result of the system simulation, the network is divided in labeled components that describe each of the classes of the input data [2]. In this section, the mathematical formulation of the LCU is reviewed considering the simplifications and considerations in [11].

A complex network can be expressed by a simple, weightless, undirected graph [10], that is, $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of vertices and $\mathcal{E}$ are the edges that connect the vertices, where $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$.

The LCU system assumes that the input network contains $|\mathcal{V}| = l + u$ vertices that are either labeled or unlabeled. The set $\mathcal{L} = \{v_1, \ldots, v_l\}$ contains the labeled vertices, where each vertex $v_i \in \mathcal{L}$ has a label $y_i \in \{1, \ldots, C\}$, with $C \ll l$. If a vertex is labeled with c, it belongs to class c. Also, the set $\mathcal{U} = \{v_{l+1}, \ldots, v_{l+u}\}$ contains the unlabeled vertices. The labeled and unlabeled are disjoint and comprise all vertices, that is, $\mathcal{L} \cap \mathcal{U} = \emptyset$ and $\mathcal{V} = \mathcal{L} \cup \mathcal{U}$.

In the system dynamics, the network is represented by the adjacency matrix $A = (a_{ij})$ where $a_{ij} = a_{ji} = 1$ if v_i is connected to v_j. The notation (i, j) refers to the edge between vertices v_i and v_j.

For practical reasons, LCU considers the following characteristics of the network:

- Most of the vertices are unlabeled, that is $l \ll u$;
- The network is connected, that is, there is only one connected component;
- At least one labeled vertex of each class is present;
- The neighborhoods are sparse, that is, $\sum_j a_{ij} \ll |\mathcal{V}|$ for every i.

Three dynamics are considered in the LCU system:

Walking. At each time, particles travel from one vertex to a neighbor one with equal probability.

Absorption. The *label domination* is the fraction of visits in an edge of class of particles that occurred at the last time. In the next movement, the transitory label domination determines the success rate of particles moving to their next vertices, or to fail and being removed from the system.

Generation. The labeled vertices act as *sources* for new particles. This rule maintains the size of the population close to its initial size.

The values of label domination assist the algorithm that solves machine learning problems. LCU unfolds the interaction network in subnetworks by group: the *unfolding* of label c at time t is the subnetwork containing all the edges dominated by label c at time t. Over the evolution of the competition process, the network's unfoldings converge to stable subnetworks that are afterward analyzed to the vertex classification task [11]. (Although more robust vertex classification rules can be devised, in this paper we consider the majority-neighborhood rule for labeling [11]).

Formally, the LCU system (with maximum competition and proper initialization [2], [11]) is

$$X(t) = \begin{bmatrix} \mathbf{n}^c(t) = \left[n_i^c(t)\right]_{i=1,\dots,|\mathcal{V}|} \\ N^c(t) = \left(n_{ij}^c(t)\right)_{i,j=1,\dots,|\mathcal{V}|} \end{bmatrix}_{c=1,\dots,C}, \quad (1)$$

where $\mathbf{n}^c(t)$ is a row vector whose elements $n_i^c(t)$ describe the expected population of particles from label c in each vertex v_i at time t. Stored in sparse matrices, the elements $n_{ij}^c(t) \in N^c(t)$ are the label domination at time t, that is, the portion of particles with label c that moved from v_i to v_j. In other words, the system counts the fraction of particles with label c at each vertex (n_i^c) and through edge (n_{ij}^c) at a given time.

The system X is a nonlinear Markovian dynamical system with the deterministic evolution function

$$\phi\colon \begin{cases} \mathbf{n}^c(t+1) = \mathbf{n}^c(t) \times P^c(X(t)) + \mathbf{g}^c(X(t)) \\ N^c(t+1) = \operatorname{diag} \mathbf{n}^c(t) \times P^c(X(t)) \end{cases}, \quad (2)$$

where $\operatorname{diag} \mathbf{v}$ is a square matrix with the elements of vector $\mathbf{v}$ on the main diagonal and $\times$ stands for the vector–matrix product. Basically, it redistributes the particles to new edges considering the previous expected amount of particles in each edge and the generation (vector $\mathbf{g}^c$) of new particles.

The function $P^c(X(t))$ gives a square matrix whose elements are

$$p_{ij}^c(X(t)) = \frac{a_{ij}}{\deg v_i} \frac{n_{ij}^c(t) + n_{ji}^c(t)}{\sum_{q=1}^{C} n_{ij}^q(t) + n_{ji}^q(t)}, \quad (3)$$

describing the probability of a particle with label c to successfully move from v_i to v_j given the label dominations on this edge. Such probability is proportional to the relative dominance of the class c to the edge.

The function $\mathbf{g}^c(X(t))$ of the system X at time t returns a row vector where the *i*-th element is

$$g_i^c(X(t)) = \rho_i^c \max\left\{0,\ 1 - \sum_{q=1}^{|\mathcal{V}|} n_q^c(t)\right\}, \quad (4)$$

and

$$\rho_i^c = \begin{cases} \dfrac{\deg v_i}{\sum_{v_j \in \mathcal{G}^c} \deg v_j} & \text{if } v_i \in \mathcal{G}^c, \\ 0 & \text{otherwise,} \end{cases} \quad (5)$$

with the set of source for particles with label c given by $\mathcal{G}^c = \{v_i | v_i \in \mathcal{L} \text{ and } y_i = c\}$. The term g_i^c represents the abundance of generated particles among the sources of the class c and keeps the total abundance of particles of class c close to 1.

Using the label domination, the subset of edges $\mathcal{E}^c(t) \subseteq \mathcal{E}$ contains the edges dominated by the particles with label c,

$$\mathcal{E}^c(t) = \left\{(i,j) \middle| \arg\max_q \left(n_{ij}^q(t) + n_{ji}^q(t)\right) = c\right\}. \quad (6)$$

The *unfolding* of network G obtained with the edges of class c at time t is $G^c(t) = (\mathcal{V}, \mathcal{E}^c(t))$.

The initial state $X(0)$ has restrictions:

$$\begin{cases} n_i^c(0) = 0 & \text{if } v_i \in \mathcal{L} \text{ and } y_i \neq c, \\ n_i^c(0) > 0 & \text{otherwise,} \end{cases} \quad (7)$$

$$\sum_i n_i^c(0) = 1, \text{ and} \quad (8)$$

$$n_{ij}^c(0) = \begin{cases} 0 & \text{if } v_i \in \mathcal{L} \text{ and } y_i \neq c, \\ 0 & \text{if } v_j \in \mathcal{L} \text{ and } y_j \neq c, \\ 1 & \text{otherwise.} \end{cases} \quad (9)$$

IV. Distributed model

The parallel approach in this work is to rewrite the original system's evolution function in order to attach time-varying quantities in all vertices and edges of the graph. If the proposed evolution rules of the quantities associated with vertices depend only on the neighbors, and the ones of the quantities associated with edges depend only on the endpoints, we can design a concurrent implementation that eliminates the need of barrier synchronization between epochs. Frameworks like Spark GraphX and Pregel can exploit such properties to pipeline fine-grained dependent operations efficiently, as shown in Figure 2.

In the original formulation of the LCU system, the generation function, Eq. (4), depends on the value of the total population of particles of each class at a given time. However, calculating this amount violates the condition aforementioned since the summation in Eq. (4) implies in a global reduction over all vertices in the graph.

Thus, this work proposes the estimator $\phi_{i,c}(t)$ which acts as a local estimator (at the vertex i) of the total particles of class c throughout the system at the time t.

In addition to the fine-grained task pipeline without barriers, as shown in Figure 3, this approach also eliminates the runtime

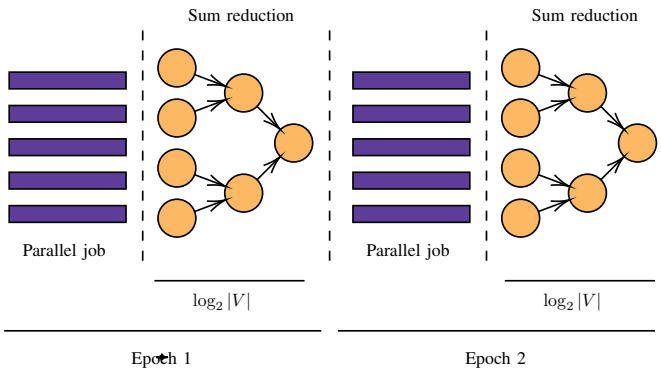

Fig. 3. Reduction effect for each epoch.

part related to the reduction over all the vertices for each epoch. It is important as we have more computational units p, once the reduction starts to dominate the runtime with minimum complexity order of $\log(|\mathcal{V}|)$, even with large p, while the other n independent and simple calculations have complexity order of n/p.

A. Proposed model

Let $\mathcal{A}_i = \{j : a_{ij} = 1\}$ be the set with the indices of the neighbors of vertex v_i, the evolution function of the system X, Eq. 2, can be rearranged as

$$n_{ij}^c(t+1) = \frac{n_i^c(t)}{\deg v_i} \frac{n_{ij}^c(t) + n_{ji}^c(t)}{\sum_{q=1}^{C} n_{ij}^q(t) + n_{ji}^q(t)}, \text{ and} \tag{10}$$

$$n_i^c(t+1) = \sum_{j \in \mathcal{A}_i} n_{ji}^c(t+1) + \rho_i^c \max\left\{0,\ 1 - \sum_{q=1}^{|\mathcal{V}|} n_q^c(t)\right\}. \tag{11}$$

In this form, it is clear that the evolution of n_i^c depends on n_q^c for all q. In order to restrict the dependencies, we propose the following system $\hat{X}$.

The variables in each vertex v_i in system $\hat{X}$ are

$$\{\hat{n}_i^c(t), \phi_i^c(t)\}_{c=1,\ldots,C}, \tag{12}$$

where $\phi_i^c(t)$ is going to estimate the summation $\sum_{q=1}^{|\mathcal{V}|} n_q^c(t)$ for every i and c at time t.

Each existing (undirected) edge ij, such that $a_{ij} = 1$, carry variables

$$\left\{\hat{n}_{ij}^c(t), \hat{n}_{ji}^c(t)\right\}_{c=1,\ldots,C}. \tag{13}$$

The proposed evolution rules of system $\hat{X}$ are

$$\hat{n}_{ij}^c(t+1) = \frac{\hat{n}_i^c(t)}{\deg v_i} \frac{\hat{n}_{ij}^c(t) + \hat{n}_{ji}^c(t)}{\sum_{q=1}^{C} \hat{n}_{ij}^q(t) + \hat{n}_{ji}^q(t)}, \tag{14}$$

$$\hat{n}_i^c(t+1) = \sum_{j \in \mathcal{A}_i} \hat{n}_{ji}^c(t+1) + \rho_i^c \max\{0,\ 1 - \phi_i^c(t)\}, \text{ and} \tag{15}$$

$$\phi_i^c(t+1) = \frac{|\mathcal{V}|\, \hat{n}_i^c(t+1) + \sum_{j \in \mathcal{A}_i} \phi_j^c(t)}{\deg v_i + 1}. \tag{16}$$

Note that the formulation of system $\hat{X}$ guarantees that the evolution of the variables in each vertex v_i depends only on itself, the neighbors v_j, and adjacent edges ij, such that $j \in \mathcal{A}_i$. In the same manner, the evolution of the variables in each existing edge ij depends only on itself and its endpoints v_i and v_j.

V. Model Validation

In this section, we validate that the proposed improvement on the LCU system does not hinder its performance on classification tasks. We also provide evidence that the estimator ϕ can successfully estimate the total abundance of particles in few iterations.

We evaluate both versions of the system, X and $\hat{X}$, in the benchmark proposed by [11]. Such benchmark comprises partially-labeled networks that represent semi-supervised classification problems with different levels of difficulty.

The main parameters of the benchmark are: q which controls the class overlap, b which controls the class border, and k which controls the average number of connection in each vertex. Increasing the values of q and b increases the difficulty of the problem. For more information about the benchmark, consult [11].

For the following experiments, wherever not specified, we fix the size of the network and the maximum number of epochs to 250 and 100, respectively, and we label randomly 10% of the vertices. Also, we repeat the experiments 30 times for each parameter configuration.

Initially, we discuss whether the proposed ϕ is a good estimator of the abundance of particles of each class. Figure 4 indicates the efficiency of our estimator. The curves in these graphs indicate the evolution of the average error between our estimator and the correct summation that we have tried to approximate over the epochs of our algorithm.

The graphs of the results showed us a very interesting result: in all situations, the ϕ values converged to the real values of the summation, arriving at errors very close to 0. Another important fact is that this convergence was very fast. In all situations, the estimators have already converged in 15 epochs of simulation.

Although the estimator approximates the real value of the summation, the errors in the first iterations could hinder the performance of the algorithm in terms of data classification. Therefore, we compare both versions of the system in terms of performance.

Figure 5 shows a comparative result between the original algorithm and the introduction of the ϕ estimator value. In the figure, we expose the average accuracy of the methods in different network configurations.

From the experiment, the accuracy rates of the methods seem similar, however, by the Wilcoxon signed rank test with continuity correction [17], we reject the hypothesis that the accuracy achieved by our method is less than or equal to the one achieved by the previous formulation (p-value 0.0314).

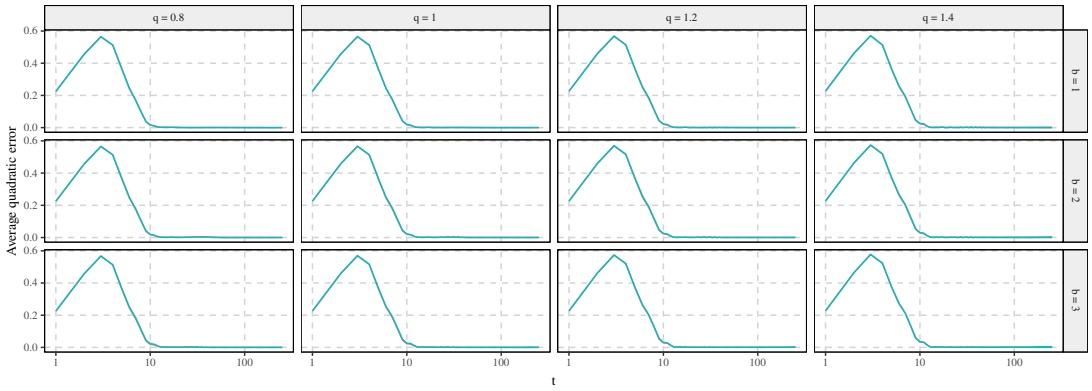

Fig. 4. Squared error of estimation during iterations and different network configurations.

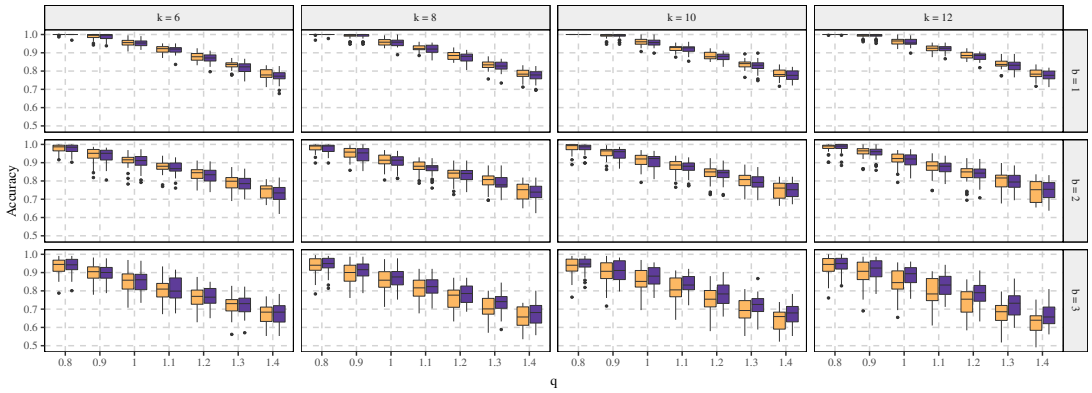

Fig. 5. Average accuracy of algorithm with (dark purple) and without (light orange) estimation ϕ applied in different levels of difficulty.

In addition to measuring the classification accuracy, we also assess the exploitation and the exploration behavior of the particles. As in [2], we measure the exploitation behavior by assessing the ability of the algorithm of identifying the border of the classes. Figure 6 plots the root mean squared error (RMSE) on predicting the purity of each vertex (consult [2] for more details of the metric).

Similar to the accuracy, using the same statistical test, we reject the hypothesis that our technique obtain higher or equal RMSE than the previous method (p-value $1.251\text{e}{-18}$).

Finally, the measure of exploration was the entropy of the particles among vertices, that is, the entropy of the class dominance in each vertex. Larger values of entropy mean that particles travel more and explore better the network, since they reach vertices far from the sources. The comparative results between the two versions of the algorithm, Figure 7. Our method explores the network a little less in easy problems, and a little more in difficult problems. By the same statistical test we reject the hypothesis that our technique reaches smaller or equal entropy compared to the other method (p-value $5.885\text{e}{-6}$).

VI. Conclusions

In this paper, we discuss an enhancement on the Labeled Component Unfolding system [2]. We propose a new formulation of its particle propagation equations, in which the summation of class dominance at each vertex is estimated. The main advantage of our formulation is that it can bypass bottlenecks produced from parallel processing [15] of the model by tools such as GraphX [16].

The proposed equation of the estimator ϕ is a good estimator since it converges to the value of the sum of the dominance in a few iterations of the system. Also, our modification presents slight better classification performance than the original formulation, as demonstrated experimentally. The increase in performance is probably due to the greater exploration behavior in the first few iterations of the system. The exploration increases because the proposed estimator ϕ underestimates the total population, and, as a consequence, more particles are generated at the beginning of the simulation. Such a phenomenon will be studied in further works.

In future works, we intend to evaluate different parallel implementations of the LCU with and without our estimator to assess the speed-up in different frameworks.

References

[1] O. Chapelle, B. Schölkopf, and A. Zien, Eds., *Semi-Supervised Learning*. Cambridge, MA: MIT Press, 2006.

[2] F. A. N. Verri, P. R. Urio, and L. Zhao, "Network unfolding map by vertex-edge dynamics modeling," *IEEE Transactions on Neural Networks and Learning Systems*, vol. PP, no. 99, pp. 1–14, 2016.

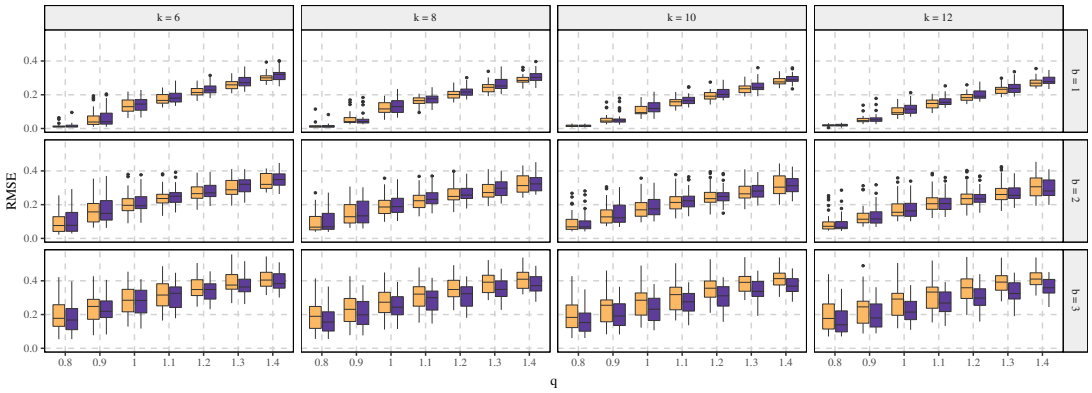

Fig. 6. Average RMSE on predicting the purity of each vertex achieved by the algorithm with (dark purple) and without (light orange) estimation ϕ applied in different levels of difficulty.

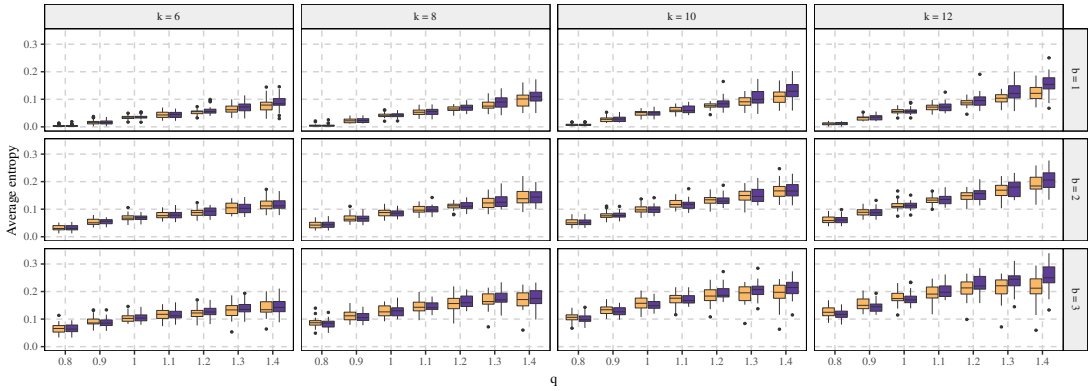

Fig. 7. Average entropy of algorithm with (dark purple) and without (light orange) estimation ϕ applied in different levels of difficulty.

[3] K. Nigam, A. K. McCallum, S. Thrun, and T. Mitchell, "Text classification from labeled and unlabeled documents using EM," *Machine Learning*, vol. 39, no. 2-3, pp. 103–134, 2000.

[4] V. N. Vapnik, *Statitical learning theory*. New York, NY: Wiley, 1998.

[5] X. Zhu and A. B. Goldberg, *Introduction to Semi-Supervised Learning*, 3rd ed. Morgan and Claypool Publishers, 2009.

[6] T. C. Silva and L. Zhao, "Network-based stochastic semisupervised learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 23, no. 3, pp. 451–66, 2012.

[7] L. Cheng and S. J. Pan, "Semi-supervised Domain Adaptation on Manifolds," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 12, pp. 2240–2249, 2014.

[8] T. C. Silva and L. Zhao, "Semi-supervised learning guided by the modularity measure in complex networks," *Neurocomputing*, vol. 78, no. 1, pp. 30–37, 2012.

[9] T. C. Silva and L. Zhao, *Machine Learning in Complex Networks*, 1st ed. Springer, 2016.

[10] M. Newman, A.-L. Barabási, and D. J. Watts, *The Structure and Dynamics of Networks: Princeton Studies in Complexity*. Princeton University Press, 2006.

[11] F. A. N. Verri, P. R. Urio, and L. Zhao, "Advantages of edge-centric collective dynamics in machine learning tasks," *Journal of Applied Nonlinear Dynamics*, vol. 7, no. 3, pp. 269–285, 2018.

[12] T. White, *Hadoop: The Definitive Guide: Storage and Analysis at Internet Scale e MapReduce: Simplified Data Processing on Large Clusters*. O'Reilly Media, Inc., 2015.

[13] H. Karau, M. Zaharia, A. Kowinski, and P. Wendell, *Learning Spark*. O'Reilly Media, Inc., 2015.

[14] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," *Proceedings of NSDI*, pp. 15–28, 2012.

[15] G. Malewicz, M. H. Austern, A. J. C. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: A system for large-scale graph processing," *Proceedings of the 2010 International Conference on Management of Data*, pp. 135–146, 2010.

[16] R. S. Xin, D. Crankshaw, A. Dave, J. E. Gonzalez, M. J. Franklin, and I. Stoica, "Graphx: Unifying data-parallel and graph-parallel analytics," *Computer Science Databases*, 2014.

[17] D. Rey and M. Neuhäuser, *Wilcoxon-Signed-Rank Test*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 1658–1659. [Online]. Available: https://doi.org/10.1007/978-3-642-04898-2_616