2018 IEEE 11th International Conference on Cloud Computing

Revolver: Vertex-centric Graph Partitioning Using Reinforcement Learning

Mohammad Hasanzadeh Mofrad, Rami Melhem
Department of Computer Science, University of Pittsburgh
Pittsburgh, USA
Email: {M.HMOFRAD, MELHEM}@pitt.edu

Mohammad Hammoud
Carnegie Mellon University in Qatar
Doha, Qatar
Email: MHHAMOUD@cmu.edu

***Abstract*—Big graph analytics is gaining a widespread momentum across different fields, including biology, computer vision, social networks, recommendation systems and transportation logistics, to mention just a few. Distributed systems for graph analytics are utilized as a mean to process big graphs. To distribute and balance computation and communication loads within a distributed graph analytics system, graph partitioning algorithms can be leveraged. In this paper, we propose Revolver, a machine learning-based graph partitioning algorithm. In particular, Revolver uses reinforcement learning and label propagation to efficiently and effectively carry out the task of graph partitioning. It employs a vertex-centric approach where each vertex in a graph is associated with an autonomous agent responsible for assigning a suitable partition to the vertex. In addition, it uses label propagation to evaluate the decency of partitioning. Evaluation results show that Revolver can produce highly balanced and localized partitions compared to three popular and state-of-the-art graph partitioning algorithms.**

Index Terms*—*Graph partitioning*; *reinforcement learning*; *learning automata*; *label propagation

I. Introduction

A myriad of real-world applications, including network traffic analysis, web search, social community detection and item recommendation, among others, can be effectively modeled as graph analytics problems. For instance, Google models the web as a graph and runs PageRank [1] on top of it to rank web pages based on users' search queries. Alongside, Facebook models its social network as a graph and utilizes EdgeRank [2] to organize users' newsfeeds. Furthermore, Amazon offers product suggestions based on items in their shopping cart using item-to-item collaborative filtering [3]. Nonetheless, before realizing and executing these kinds of large-scale graph analytics applications, a certain preprocessing step, namely, *graph partitioning* is necessitated. Graph partitioning involves partitioning or overpartitioning an input (big) graph into smaller subgraphs, which are subsequently distributed and processed (e.g., using a distributed graph analytics system like Google's Pregel [4] or Apache Giraph [5]) across a cluster of machines.

Among the most popular graph partitioning algorithms are Random and Hash partitioning. These two algorithms are very simple and do not require global views of input graphs while partitioning, hence, can scale extremely well with big graphs. However, they result in poor data locality and, accordingly, high communication overhead. To this end, a major objective of any new suggested graph partitioning algorithm is to outstrip Random and Hash partitioning via improving data locality and reducing communication traffic, while maintaining balance across all partitions.

It has been more than a semicentennial that Arthur Lee Samuel coined the name Machine Learning (ML) in 1959 [6]. Since then, ML has influenced a vast array of Computer Science developments and an increasingly large swath of industries. Reinforcement Learning (RL) [7] is a class of ML algorithms inspired by the stimulus-response principle, which is concerned with how an autonomous agent can take an action in an interactive environment so as to optimize a cumulative reward. The most sophisticated RL agent thus far is the Deep Q-network. It combines Deep Neural Networks (a class of ML algorithms based on learning data representations) with RL. This agent was capable of playing a diverse range of Atari 2600 games [8]. Learning Automata (LA) [9] are a subclass of RL algorithms, which select their new actions using their past experiences with certain environments. Despite their simple structure, LA are useful for control learning and have applications in Evolutionary Computation [10], [11], Cloud and Grid computing [12], [13], Social Networks [14], Complex Networks [15], Image Processing [16], and data clustering [17].

In this paper, we propose Revolver, a graph partitioning algorithm that trains LA to partition a graph. It adopts a vertex-centric view of the graph where a learning automaton is assigned to each vertex. Afterwards, the learning automaton carries out the task of determining a partition label to its associated vertex by using its limited knowledge collected from its neighbors. Revolver uses Label Propagation (LP) [18] to evaluate the partitioning assignments of LA and provide them with proper feedback to refine the overall partitioning quality. By pursuing several rounds of actions - feedback, Revolver attempts to balance the partitions in terms of the number of edges contained in each partition.

The paper is organized as follows. Section II defines the problem and provides a background on LA and LP. In Section III, we share the core idea of Revolver and demonstrate how LA and LP can be leveraged to craft a graph partitioner. Section IV presents the experimental results. Finally, the paper is concluded in Section V.

2159-6190/18/$31.00 ©2018 IEEE
DOI 10.1109/CLOUD.2018.00111

II. Background

A. Problem Statement

Given a graph $G = (V, E)$ where V is the set of vertices and E is the set of edges, k-way balanced partitioning divides the graph into k subgraphs

$$\frac{|E|}{k} \cdot (1 + \epsilon),\ \epsilon > 0 \tag{1}$$

where $\epsilon > 0$ is the imbalanced ratio. Here, we consider edges rather than vertices as atomic units of computation. Hence, the graph is partitioned evenly based on the number of edges $|E|$ and not the number of vertices $|V|$, especially that natural graphs follow power-law degree distribution where a subset of vertices owns a huge portion of edges.

B. Learning Automata

Learning automaton [9] is a probabilistic algorithm, which belongs to the family of RL algorithms. It is adaptive, wherein actions are selected based on a specific probability distribution. A selected action is applied to the surrounding environment of the learning automaton and a feedback is provided back to the automaton based on the merit of changes. Through multiple episodes of action selections and response receptions from the environment, the learning automaton eventually learns an optimal action. The variable structure learning automaton, which is the conventional and common type of learning automaton, is defined using the quadruple $[A, P, R, T]$ where $A = \{a_1, ..., a_m\}$ is the set of actions with m being the number of actions, $P = \{p_1, ..., p_m\}$ is the probability vector where $\sum_{i=1}^{m} p_i = 1$, $R \in \{0, 1\}$ is the set of reinforcement signals, and T is the linear learning algorithm where in n^{th} step $P(n + 1) = T[A(n), P(n), R(n)]$.

In the n^{th} step of T, if action $a_i(n)$ receives reward signal $R(n) = 0$, the associated probability vector $P(n)$ is updated using (2), otherwise, if $a_i(n)$ receives penalty signal $R(n) = 1$, $P(n)$ is updated using (3)

$$p_j(n+1) = \begin{cases} p_j(n) + \alpha \cdot (1 - p_j(n)) & j = i \\ p_j(n)(1 - \alpha) & j \neq i \end{cases} \tag{2}$$

$$p_j(n+1) = \begin{cases} p_j(n)(1 - \beta) & j = i \\ p_j(n)(1 - \beta) + \frac{\beta}{m-1} & j \neq i \end{cases} \tag{3}$$

where in (2) and (3), α and β are reward and penalty parameters, respectively.

C. Label Propagation

To partition a big graph, a scalable algorithm needs to make a *local* decision per each vertex (assuming a vertex-centric approach) based on the vertex's neighborhood rather than a *global* decision based on the whole graph, which may consume tremendous time. The distributed characteristic and connectivity layout of graphs make them naturally suitable for algorithms like LP, which relies on local information. For instance, Spinner [19] (which is the closest algorithm to Revolver) proposes an LP-based algorithm to solve k-way graph partitioning. The utilized LP is simply a scoring function, which produces a score for each partition label of a vertex using its neighboring vertices. More information about Spinner can be found at [19].

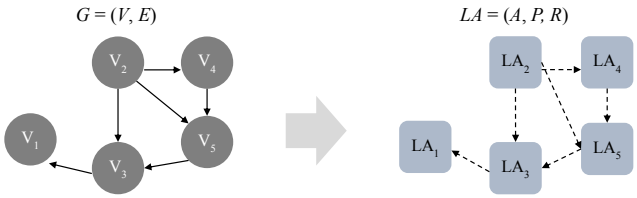


Fig. 1. A network of LA is allocated for nodes of a graph.

III. Revolver

Revolver is an existing manifestation of reinforcement learning aimed at performing high-quality graph partitioning. More precisely, it uses LA and LP for partitioning (big) graphs. In Revolver, LP is extrapolated to form an objective function, which emits a signal to measure the decency of the assigned labels. We now elaborate on Revolver.

A. Normalized Label Propagation

Generally, the LP algorithm produces a score for each available partition of a vertex. The LP algorithm introduced in Spinner [19] constitutes of a weight and a penalty terms. The penalty term controls the load balancing of edges, yet it is not normalized by the total number of edges. In addition, the produced score is not normalized by the weight and penalty terms. To address these critical flaws, Revolver utilizes a normalized LP defined as follows:

$$score(v, l) = \frac{1}{2}\Big(\sum_{u \in N(v)} \frac{w(u, v) \cdot \delta(\psi(u), l)}{\sum_{u \in N(v)} w(u, v)} - \frac{1 - (b(l)/C)}{\sum_{i=1}^{k} 1 - (b(l_i)/C)} \Big) \tag{4}$$

where in (4), the first term (weight term) is normalized based on the total weight of v's neighborhood, $N(v)$, and the second term (penalty term) is normalized based on the total load of the system. Also, δ is the Kronecker delta, w is the weighting function, ψ is the labeling function, $b(l)$ is the load of l^{th} partition, and C is the capacity of the partition.

B. Transforming the graph partitioning problem into the reinforcement learning axiom

To partition a graph $G = (V, E)$ into k partitions using reinforcement learning, $LA = (A, P, R)$ needs to be well-architected. With a vertex-centric approach, it is intuitive to assign a learning automaton for each vertex, after which the available range of partitions can constitute the learning automaton action set. This mapping is shown in Figure 1 and enables a network of LA to partition the vertices. In particular, the way LA can be laid-out for graph partitioning is as follows: 1) a network of LA is created where $|V| = |LA|$, 2) a learning automaton can find neighboring LA using the

set E, 3) the action set of a learning automaton is the same as the range of available partitions $|A| = k$, 4) a learning automaton determines the partition for its associated vertex using its action set, 5) a reinforcement signal is required to evaluate the merit of the partitioning configuration and provide the learning automaton with a feedback, and 6) the network of LA will learn how to perform the task of graph partitioning after a series of actions and feedback receptions.

Lastly, we note that we used a new specialized version of learning automaton algorithm, which collectively applies the reinforcement signals on the entire action set using weights relative to the calculated scores. Due to a space constraint, we did not report this new specialized version here.

C. Using label propagation to train a learning automaton

Now that the key idea of Revolver is grounded, it is time to show how LP is used to train the learning automata algorithm.

1) Action selection: Each learning automaton stochastically takes an action based on its probability distribution and selects a candidate partition to migrate to. The more the probability of an action, the higher the chance it might be selected.

2) LP scoring function: The scoring function (4) produces a score for each label/partition of vertex v based on the partitions of v's neighboring vertices and the load of current v's partition.

3) Vertex migration: The probability of migrating vertex v to a new candidate partition is calculated using v's current load and the remaining available load of the partition. If the candidate partition selected by the learning automaton is different from v's current partition, v migrates to the candidate partition with a specific probability of migration as in [19].

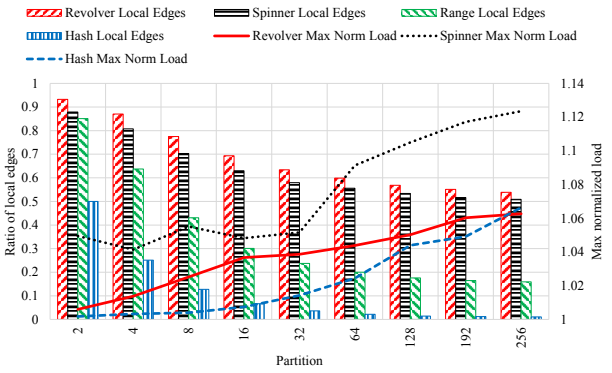
4) Calculating the reinforcement signal: For each learning automaton associated with vertex v, a reinforcement signal R is calculated as follows: i) the learning automaton will receive the reward signal ($R = 0$), if the selected action $a_{\psi(v)}$ has the highest score in (4) or the probability of migrating to the candidate partition is more than zero; ii) otherwise, the learning automaton will receive the penalty signal ($R = 1$).

5) Updating probability vector: After calculating the reinforcement signal R for each learning automaton, the probability vector is updated using (2) or (3).

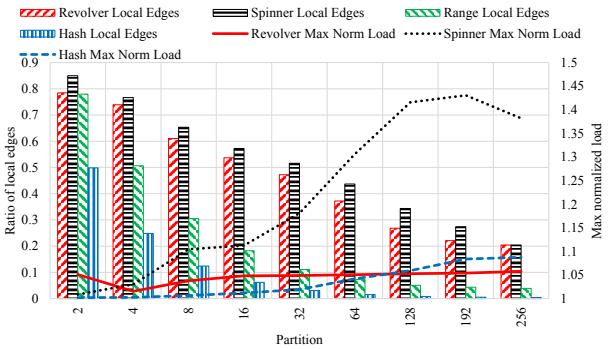
IV. Experiments

A. Experimental Setup and Metrics

We compare Revolver against Spinner [19], which is the state-of-the-art LP-based graph partitioning algorithm, and Range and Hash partitioning, which are two popular partitioning algorithms. As in [19], we measure the quality of partitioning via the following two metrics: $Ratio\ of\ local\ edges = \frac{|local\ edges|}{|E|}$ and $Max\ normalized\ load = \frac{|max\ load|}{|E|/k}$. Here, a *local edge* is an edge that has both of its ends located at the same partition and $|max\ load|$ is the number of edges in the highest loaded partition. Clearly, the *Ratio of local edges* captures the locality in the produced partitions, while the *Max normalized load* defines the amount of computation inside the highest loaded partition.



(A) LiveJournal



(B) Higgs-twitter (TWIT)

Fig. 2. Average *Ratio of local edges* and *Max normalized load* of Revolver, Spinner, Range and Hash over 10 individual runs

As for datasets, we use: 1) LiveJournal (LJ) with 4.84 million vertices and 68.99 million edges, and 2) Higgs-twitter (TWIT) with 0.45 million vertices and 14.85 million edges. Lastly, we configure Spinner as in [19] and set the learning automaton reward and penalty parameters α and β to 1 and 0.1, respectively.

B. Experimental Results

Figure 2 shows the *Ratio of local edges* and *Max normalized load* for 10 individual runs of Revolver, Spinner, Range and Hash algorithms across different numbers of partitions 2, 4, 8, 16, 32, 64, 128, 192, and 256. We note that the *Max normalized load* of Range partitioning was extremely inferior (e.g. on TWIT, Revolver achieved 38x improvement versus Range partitioning with 256 partitions), hence, it was excluded from Figure 2.

On LJ (Figure 2(A)), Revolver produces the best *Ratio of local edges* across all partitions, whereas Hash partitioning results in better *Max normalized load*. Nonetheless, Revolver meets (and even outperforms) Hash with 256 partitions, indicating that it might even yield better results if the number of partitions is increased further.

Results for TWIT (Figure 2(B)) show that Revolver can efficiently utilize the extra capacity and produce balanced partitions, while in contrary Spinner produces imbalanced ones. Note that Spinner produces the best *Ratio of local edges*, but evidently at a huge cost of highly imbalanced partitions.

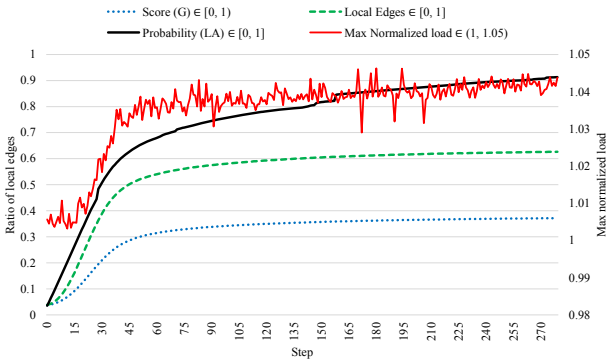


Fig. 3. Convergence characteristics of Revolver on LJ with 32 partitions.

In Figure 2, Hash partitioning has the worst *Ratio of local edges* across all partitions, however, it can generate balanced partitions, yet only for a small number of partitions (2 - 64). This observation does not give Hash a privilege because it over-utilizes the 5% extra capacity ($\frac{|E|}{k}(1+\epsilon) \mid \epsilon = 0.05$) for larger partitions. Moreover, Hash failed to distribute the load on larger number of partitions (128 - 256) for LJ and TWIT. In contrast, Spinner exceeds the extra capacity on both graphs, particularly when the number of partitions is increased.

As shown in Figure 2(A-B), Revolver improves *Max normalized load* by up to 6% and 20% compared to Spinner using LJ and TWIT, respectively. These results show that Revolver is a scalable algorithm that can always maintain load balance across all partitions. Moreover, it enhances the *Ratio of local edges* by up to 9% on LJ and yields comparable results on TWIT as opposed to Spinner. Such results verify that Revolver partitions are more localized and have smaller edge cuts.

C. Convergence Test of Revolver

Figure 3 shows the evolution process of Revolver on LJ with 32 partitions. For a graph $G = (E, V)$, $score(G) = \sum_{v \in V} score(v)/|V|$, where $score(v) \in [0, 1]$ is computed using (4) (the LP scoring function). Also, for a LA = (A, P, R), $probability(LA) = \sum_{la \in LA}(P(\psi(la)) > 0.9)/|V|$, where $probability(LA) \in [0, 1]$ is the number of actions with probabilities greater than a threshold of 0.9 divided by the number of vertices.

To this end, the *Ratio of local edges* and *Max normalized load* metrics in Figure 3 keep improving while the probability of LA increases. The left shoulder of the plot corroborates this behavior, especially after 50 steps when the probability of LA passes 0.7 and LA decisions render more precisely (considering probability = 0.5 as random decision making). Alongside, the *Max normalized load* (right y-axis of Figure 3) catches up after 50 steps and allows Revolver to efficiently utilize the extra 5% capacity (c = 0.05). Finally, the probability converges to almost 0.95 after passing 270 steps.

V. Conclusions

In this paper, we proposed Revolver, a partitioning algorithm that partitions (big) graphs using reinforcement learning. In Revolver, each vertex is armed with a learning automaton, which determines the vertex partition. Revolver leverages label propagation to evaluate the partitioning configuration and provide feedback to learning automata. Our experiments showed that Revolver is scalable and can maintain an excellent ratio of local edges to balanced edges compared to other popular and state-of-the-art algorithms.

VI. Acknowledgments

This publication was made possible by NPRP grant #7-1330-2-483 from the Qatar National Research Fund (a member of Qatar Foundation). The statements made herein are solely the responsibility of the authors. This research was supported in part by the University of Pittsburgh Center for Research Computing through the resources provided.

References

[1] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, Tech. Rep., 1999.
[2] N. B. Ellison, J. Vitak, R. Gray, and C. Lampe, "Cultivating social resources on social network sites: Facebook relationship maintenance behaviors and their role in social capital processes," *Journal of Computer-Mediated Communication*, vol. 19, no. 4, pp. 855–870, 2014.
[3] G. Linden, B. Smith, and J. York, "Amazon. com recommendations: Item-to-item collaborative filtering," *IEEE Internet computing*, vol. 7, no. 1, pp. 76–80, 2003.
[4] 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 *2010 ACM SIGMOD International Conference on Management of data*, 2010, pp. 135–146.
[5] C. Martella, R. Shaposhnik, D. Logothetis, and S. Harenberg, *Practical graph analytics with apache giraph*. Springer, 2015.
[6] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of research and development*, vol. 44, no. 1.2, pp. 206–226, 2000.
[7] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1, no. 1.
[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
[9] K. S. Narendra and M. A. Thathachar, *Learning automata: an introduction*. Courier Corporation, 2012.
[10] M. Hasanzadeh, M. R. Meybodi, and M. M. Ebadzadeh, "Adaptive cooperative particle swarm optimizer," *Applied Intelligence*, vol. 39, no. 2, pp. 397–420, 2013.
[11] M. Hasanzadeh, S. Sadeghi, A. Rezvanian, and M. R. Meybodi, "Success rate group search optimiser," *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 28, no. 1-2, pp. 53–69, 2016.
[12] M. Hasanzadeh and M. R. Meybodi, "Grid resource discovery based on distributed learning automata," *Computing*, vol. 96, pp. 909–922, 2014.
[13] M. Hasanzadeh and M. R. Meybodi, "Distributed optimization grid resource discovery," *The Journal of Supercomputing*, vol. 71, no. 1, pp. 87–120, 2015.
[14] A. Rezvanian and M. R. Meybodi, "Sampling social networks using shortest paths," *Physica A: Statistical Mechanics and its Applications*, vol. 424, pp. 254–268, 2015.
[15] A. Rezvanian, M. Rahmati, and M. R. Meybodi, "Sampling from complex networks using distributed learning automata," *Physica A: Statistical Mechanics and its Applications*, vol. 396, pp. 224–234, 2014.
[16] M. H. Mofrad, S. Sadeghi, A. Rezvanian, and M. R. Meybodi, "Cellular edge detection: Combining cellular automata and cellular learning automata," *AEU-International Journal of Electronics and Communications*, vol. 69, no. 9, pp. 1282–1290, 2015.
[17] M. Hasanzadeh-Mofrad and A. Rezvanian, "Learning automata clustering," *Journal of Computational Science*, vol. 24, pp. 379–388, 2018.
[18] U. N. Raghavan, R. Albert, and S. Kumara, "Near linear time algorithm to detect community structures in large-scale networks," *Physical review E*, vol. 76, no. 3, p. 036106, 2007.
[19] C. Martella, D. Logothetis, A. Loukas, and G. Siganos, "Spinner: Scalable graph partitioning in the cloud," in *ICDE*, 2017, pp. 1083–1094.