ISSN 1995-0802, Lobachevskii Journal of Mathematics, 2018, Vol. 39, No. 9, pp. 1262–1269. © Pleiades Publishing, Ltd., 2018.

Unsupervised Graph Anomaly Detection Algorithms Implemented in Apache Spark

A. Semenov[1*], A. Mazeev[1], D. Dоropheev[2***], and T. Yusubaliev[3****]**

(Submitted by V. V. Voevodin)

[1]Scientific Research Centre for Electronic Computer Technology (NICEVT) JSC, Varshavskoe sh. 125, Moscow, 117587 Russia

[2]Moscow Institute of Physics and Technology (State University), Institutskii per. 9, Dolgoprudny, Moscow oblast, 141701 Russia

[3]Quality Software Solutions Ltd., Moscow, Russia

Received June 28, 2018

Abstract—The graph anomaly detection problem occurs in many application areas and can be solved by spotting outliers in unstructured collections of multi-dimensional data points, which can be obtained by graph analysis algorithms. We implement the algorithm for the small community analysis and the approximate LOF algorithm based on Locality-Sensitive Hashing, apply the algorithms to a real world graph and evaluate scalability of the algorithms. We use Apache Spark as one of the most popular Big Data frameworks.

DOI: 10.1134/S1995080218090184

Keywords and phrases: *Spark, graph processing, supervised anomaly detection, performance evaluation.*

1. INTRODUCTION

In recent years, data intensive applications have become widespread and appeared in many science and engineering areas. They are characterized by a large amount of data, irregular workloads, unbalanced computations and low sustained performance of computing systems. Development of new algorithmic approaches and programming technologies are urgently needed to boost efficiency of HPC systems for similar applications thus enabling advancing of HPC and Big Data convergence [1].

Anomaly detection in graphs occurs in many application areas, for example, in the analysis of financial markets, in spam filtering, as well as in detection of cyber attacks. Often graphs are huge and require high performance processing and more than one node of a cluster for a memory reason.

Spark [2] is a framework which optimizes programming and execution models of MapReduce [3] by introducing a resilient distributed dataset (RDD) abstraction. Users can choose between the cost of storing RDD, the speed of accessing it, the probability of losing part of it, and the cost of recomputing it. Apache Spark is a popular open-source implementation of Spark. It supports a rich set of high-level tools including MLlib for machine learning and GraphX for graph processing.

In this research we have implement using Apache Spark two unsupervised anomaly detection algorithms, apply the algorithms to a real graph and evaluate their performance.

[*]E-mail: semenov@nicevt.ru
[**]E-mail: mav367@mail.ru
[***]E-mail: dmitry@dorofeev.su
[****]E-mail: ytr@kpr-it.com

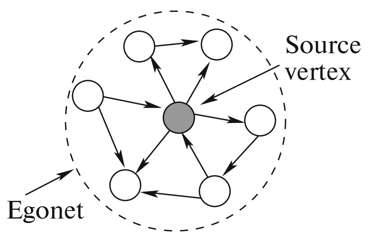

Fig. 1. Example of egonet with radius is equal to 1.

2. ANALYSIS

We consider a directed weighted graph. Anomaly is an object (e.g. vertex, edge or subgraph) in a graph that differs the most other objects of the same type in the graph by a set of features. We have a real world graph from the e-commerce application area and we consider the problem of searching edges as anomalies.

The authors of the survey [4] outline that the graph anomaly detection problem can be solved by spotting outliers and anomalies in unstructured collections of multi-dimensional data points, which can be obtained by graph analysis algorithms. The algorithms transform the graph anomaly detection problem to the well-known and understood outlier detection problem.

The following groups of graph features can be outlined:

- Vertex-level features (degrees, attributes, centrality measures, etc.);
- Egonet-level features (number of triangles, total weight, etc.);
- Vertex-group-level features (measures of vertex group: density, modularity, etc.);
- Global features (number of connected components, distribution of component sizes, minimum spanning tree weight, etc.).

We consider vertex-level and egonet-level features. An egonet is the neighborhood with radius 1 around a vertex; including the vertex, its direct neighbors and all the edges among these vertices. Fig. 1 shows an egonet example.

In our application area we consider group level patterns. In [5] the authors introduce blackhole and volcano patterns. A blackhole pattern is a community of vertices that has only incoming edges to the community vertices from the other vertices in the graph, a volcano pattern is a community of vertices that has only outgoing edges to the community vertices from the other vertices in the graph. But in the current research we do not implement any volcano and blackhole search algorithm.

We consider two algorithms for the anomaly detection problem: OddBall and LOF. The OddBall algorithm [6] is based on an egonet extraction. It calculates egonet-based features and finds patterns that most of the egonets of the graph follow with respect to those features. We apply the method to spot anomalous egonets and hence anomalous edges.

The Local Outlier Factor (LOF) method [7] is based on local normalized density between k nearest-neighbor points. This normalization is a key to addressing challenges with varying cluster density in data. The main problem in the LOF method is the nearest neighbors search for every point. There are different methods to the nearest neighbors search. In the naive accurate brute-force algorithm [8] for every point one can calculate distances to all other points and sort the values, then for every point k best points are taken. The brute-force algorithm is too time-consuming for a large data set. In accurate approach with space partitioning [9] an n-dimension space is splitted apart, but space partitioning algorithms are slower than the brute-force algorithm when the number of dimensions exceeds 10 [9]. There are a number of other techniques and data structures for the nearest neighbors search: cover tree [11], clustering [13].

There are a lot of methods for the approximate nearest neighbors search. They can help to reduce calculation time but we need to control error value in a permission range. The LSH (Locality-Sensitive

Hashing) approach [10] is one of the most efficient and popular approaches for high-dimensional nearest neighbors search [14].

The LSH approach relies on a locality-preserving hash function, it creates several hash tables that hash together similar points with high probability. The less distance between a pair of points, the more probability of a collision. In such a manner the hash table contains a relatively small set of objects which are good candidates to be the nearest to the query point. Potentially, one can obtain a very good approximate solution to the similarity search [15].

Typically, scientists use a large number of hash tables and apply different techniques to address the problem of non-uniform distribution of data [16]. We try to apply these techniques in the implementation.

3. IMPLEMENTATION

We consider the problem of detection of anomaly edges in graph. At first, it is necessary to calculate features for each edge of a graph. We consider every vector of features as a point in the n-dimension space and apply the developed algorithms.

We use the following features for each edge (26 features total): a set of features for the source vertex, the destination vertex and weight. The set of features for a vertex:

- Degree, in-degree, out-degree, and the number of vertices in the vertex's egonet with radius 1;
- Average degree, in-degree, out-degree in the vertex egonet with radius 1;
- Number of vertices-volcanoes (vertices with null in-degree), vertices-blackholes (vertices with null out-degree) and number of other vertices;
- Number and the total weight of incoming and outgoing edges in the vertex egonet with radius 1.

The feature set is the same for OddBall and LOF algorithms.

3.1. OddBall

OddBall method considers the following egonet features:

- Number of vertices in the egonet;
- Number of edges in the egonet.

Suppose that an i-th egonet consists of N_i vertices and E_i edges. If $\alpha < \log_{N_i} E_i < \beta$, then the i vertex is normal, else—an anomaly object, where α and β are constants (float numbers), that should be defined on the evaluation stage.

The algorithm also takes into consideration the structure of the graph. Algorithms for anomaly detection in graphs often based on community extraction, but usually they extract community with large radius. The OddBall algorithm considers only communities with radius 1.

Table 1. System configuration of the Angara-cluster

Server	SuperServer 5017GR-TF
Processor	E5-2660 (8 cores, 2.2 GHz)
Memory	DDR3 64 GB
Number of nodes	36
Interconnect	Angara 4D-torus $3 \times 3 \times 2 \times 2$
	1 Gbit/s Ethernet
Operating system	SLES 11 SP4
Spark	Apache Spark 2.1.1
Scala Compiler	sbt 0.13.13

3.2. Approximate LOF

LOF (Local Outlier Factor) is an outlier detection algorithm that calculates a value for every point. With calculated value we can understand which points are anomalies. If LOF value for a point is close to 1 then point considered as a normal, else as an anomaly. A concrete threshold should be chosen by hands after data analysis.

We denote by $k\_distance(A)$ the distance from a point A to the k-th nearest neighbor. We denote by $N_k(A)$ k nearest neighbors of the A point, $d(A, B)$ is distance between the A and B points.

Denote $reachability\_distance$:

$$reachability\_distance_k(A, B) = max(k\_distance(B), d(A, B)).$$

Denote $local\_reachability\_density$ ($lrd(A)$):

$$lrd(A) = 1/\left(\frac{\sum_{B\in N_k(A)} reachability\_distance_k(A, B)}{|N_k(A)|}\right).$$

Eventually lrd value of a point compared with lrd values of the point neighbors:

$$LOF_k(A) = \frac{\sum_{B\in N_k(A)} \frac{lrd(B)}{lrd(A)}}{|N_k(A)|}.$$

We have developed a parallel approximate LOF implementation based on the LSH approach. The main problem in the LOF method is the nearest neighbors search for every point.

The scheme of the developed approximate nearest neighbors search based on the LSH approach is showed in Algorithm 1. The main idea of the LSH approach for the nearest neighbors search is for every point we calculate hash and the less distance between a pair of points the more probability of a collision. We create a series of tables ($NumTables = 100$) and random vectors for every table. For the first table we create a number of random vectors ($StartNumVectors = 3$), for each subsequent table we increment the number of random vectors by a one. Each random vector consists of d random values with Gaussian distribution with mean 0. The hash calculated for each point is a bit sequence where i-th bit is 1 if dot product of i-th random vector of a table and the point more or equal 0, and 0 in other cases.

Then we take cells of hash tables which size is less than a *const* ($4 * kNN$, kNN is number of k nearest neighbors). For each point of the cell we obtain other points of the cell as candidates of the point and add the candidates to the result for the point. For every point in the result array we remove duplicates and select best candidates (kNN maximum).

After the last iteration we take all cells with size more than the *const* and for every point in the cells we take kNN random candidates from the cell, and then erase duplicates and select the best candidates (maximum kNN). We take random candidates for every point from the same cell because when the number of random vectors is large, the cell corresponds to the small part of the n-dimension space and all the points from the cell are very close to each other. At the end for the points with less than

kNN neighbors we use brute-force algorithm because the number of the points is very small. We fix $kNN = 10$ in the implementation.

```
Input: points—set of all points
Output: result—map that storage nearest neighbors list forevery point
result = ∅
nv = Start Num Vectors
//create hash table
hash_table = ∅
for 1..Num Tables do
    //clear hash table every iteration
    hash_table = ∅
    hash_vecs = get_random_vecs(nv)
    forall the p ∈ points do
        hash = get_hash(hash_vecs, p)
        hash_table[hash]+ = p
    end
    foreach cell ∈ hash_table, cell.size < const do
        forall the point ∈ cell do
            result[point]+ = all points ∈ cell without the point
        end
    end
    forall the point ∈ result do
        result[point] = save_only_best_kNN_neighbors
    end
    nv = nv + 1
end
foreach cell ∈ hash_table, cell.size > const do
    forall the point ∈ cell do
        result[point] = save_any_kNN_neighbors ∈ cell
    end
end
forall the point ∈ result do
    result[point] = save_only_best_kNN_neighbors
end
foreach point : result[point].size < kNN do
    result[point]= brute_force(point)
end                                                        (1)
```

Algorithm 1: Nearest neighbors search algorithm based on the LSH approach.

We implement the algorithms using resilient distributed dataset API (RDD [2]). The latest Spark program interface DataFrame [19] seems to be more efficient, we plan to use it in the future work.

4. EVALUATION

Implementation evaluation was done on the Angara-C1 cluster in JSC NICEVT. The cluster has 36 nodes, but we consider only 8 in the current paper. The cluster configuration is shown in Table 1. The cluster is equipped with the Angara interconnect [20], but for the evaluation we use 1 Gigabit/s Ethernet.

4.1. Considered Graphs

We use a real graph G_r from the e-commerce application area with $|V| = 114791$ vertices and $|E| = 781440$ edges. For performance evaluation we also use synthetic Erdos–Renyi graphs [21].

Table 2. Execution time of the algorithms on the G_r graph

Feature calculation	OddBall	LOF
107.66	8.03	1978.79

Table 3. Quality results evaluation of approximate LOF algorithm compared with accurate LOF for 6000 data points

THR	FN/P	FP/N
2.5	8.03%	1.2%
3	10.27%	0.9%
3.5	12.13%	0.8%

4.2. Results

We run OddBall and LOF algorithms for the G_r graph and take 5% edges with the biggest values obtained by the OddBall algorithm and the same number of edges with the biggest LOF values. Anomaly edge part (5%) corresponds to the 2.05 threshold LOF value. Then we compare the sets: each method finds 39072 anomaly edges (5% of 781440 edges in the G_r graph), intersection of the sets is 1492 edges—3.82%. It is very interesting that two algorithms for the same feature set spot different edges as anomalies.

Table 2 contains execution time of the algorithms on 8 nodes of the cluster for the G_r graph. Execution time of the LOF algorithm does not include time of the feature calculation stage.

The quality of the LOF algorithm based on the LSH approach is shown in Table 3. THR—threshold for anomaly detection. If and only if $LOF(a\_point) > THR$ then the point is the anomaly object. FN—false-negative objects, i.e. anomalies detected as normal points by approximate method. FP—false-positive objects, i.e. normal objects detected as anomalies by approximate method. P—number of anomalies, N—number of normal objects. The accurate LOF results were obtained on random 6000 data points from the G_r graph by the brute-force algorithm.

4.3. Scalability Issues

Figure 2 shows strong scaling of the feature calculation stage, LOF based on the LSH method, Fig. 3 shows their speedup. We use 8 cores on each computational node of the cluster. The implementations show good scalability.

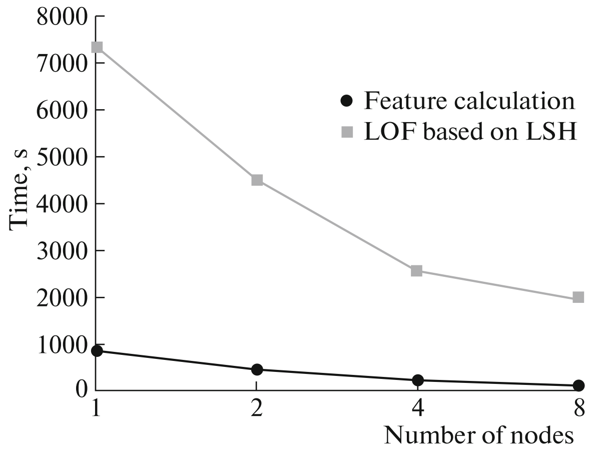

Fig. 2. Strong scaling. Features calculation for Erdos–Renyi graph with 2^{19} vertices and 2^{22} edges. Approximate LOF is on the G_r graph.

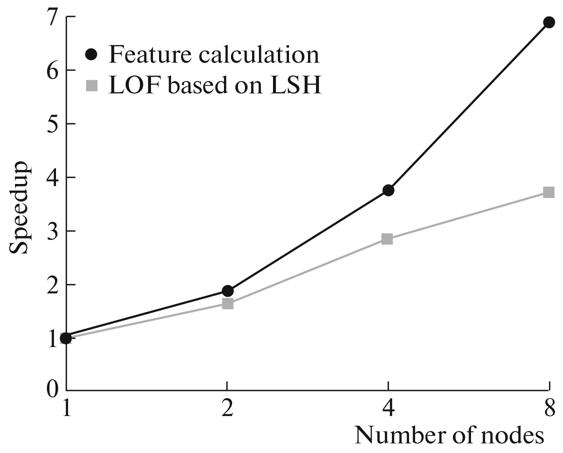

Fig. 3. Speedup. Features calculation for Erdos–Renyi graph with 2^{19} vertices and 2^{22} edges. Approximate LOF is on the G_r graph.

5. CONCLUSION

In this paper we have considered the graph anomaly detection problem. The problem was to find anomalies in a real world graph and to obtain a scalable implementation using the Apache Spark framework.

We use approach of spotting outliers in unstructured collections of multi-dimensional data points, which can be obtained by graph analysis algorithms. We implement the OddBall algorithm for the egonet analysis and the approximate LOF algorithm based on the LSH. For implementation we use the Apache Spark framework. We apply the algorithms to the real world graph and obtain different sets of edges as anomalies by each algorithm. The algorithms show good scalability on the 8-node cluster.

Research is being conducted with the finance support of the Ministry of Education and Science of the Russian Federation Unique ID for Applied Scientific Research (project) RFMEFI57816X0218. The data presented, the statements made, and the views expressed are solely the responsibility of the authors.

REFERENCES

1. D. Reed and J. Dongarra, "Exascale computing and big data: the next frontier," Commun. ACM **57** (7), 56–68 (2014).
2. M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets. HotCloud," in *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing, 2010,* pp. 10–10. http://static.usenix.org/legacy/events/hotcloud10/tech/full_papers/Zaharia.pdf. Accessed 2018.
3. J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," Commun. ACM **51**, 107–113 (2010).
4. L. Akoglu, H. Tong, and D. Koutra, "Graph based anomaly detection and description: a survey," Data Min. Knowl. Disc1ov **29**, 626–688 (2015). https://arxiv.org/pdf/1404.4679.pdf. Accessed 2018.
5. Z. Li, H. Xiong, and Y. Liu, "Detecting blackholes and volcanoes in directed networks," arXiv:1005.2179 (2010). https://arxiv.org/pdf/1005.2179.pdf. Accessed 2018.
6. L. Akoglu, M. McGlohon, and C. Faloutsos, "OddBall: Spotting Anomalies in Weighted Graphs," in *Proceedings of the 14th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining PAKDD'10, 2010,* Part 2, pp. 410–421. http://repository.cmu.edu/cgi/viewcontent.-cgi?article=3599&context=compsci. Accessed 2018.
7. M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "LOF: identifying density-based local outliers," in *Proceedings of the ACM SIGMOD 2000 International Conference on Management of Data, 2010.* http://www.dbs.ifi.lmu.de/Publikationen/Papers/LOF.pdf. Accessed 2018.
8. R. Weber, H.-J. Schek, and S. Blott, "A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces," in *Proceedings of the 24rd International Conference on Very Large Data Bases* (Morgan Kaufmann, 1998), pp. 194–205.
9. D. T. Lee and C. K. Wong, "Worst-case analysis for region and partial region searches in multidimensional binary search trees and balanced quad trees," Acta Inf. **9**, 23–29 (1977).

10. H. Koga, T. Ishibashi, and T. Watanabe, "Fast agglomerative hierarchical clustering algorithm using locality-sensitive hashing," Knowledge Inf. Syst. **12**, 25–53 (2007).
11. A. Beygelzimer, S. Kakade, and J. Langford, "Cover trees for nearest neighbor," in *Proceedings of the 23rd International Conference on Machine Learning, 2006,* pp. 97–104. https://homes.cs.washington.-edu/ sham/papers/ml/cover_tree.pdf. Accessed 2018.
12. R. Weber and S. Blott, "An approximation-based data structure for similarity search," Technical Report No. 24, ESPRIT Project HERMES No. 9141 (1997).
13. S. Ramaswamy and K. Rose, "Adaptive cluster-distance bounding for nearest neighbor search in image databases," IEEE Int. Conf. Image Process. **6**, 381–384 (2007). http://citeseerx.ist.psu.edu/viewdoc/-download?doi=10.1.1.80.6562&rep=rep1&type=pdf.
14. Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, "Multi-probe LSH: efficient indexing for high-dimensional similarity search," in *Proceedings of the VLDB Conference, 2007,* pp. 950–961. http://www.cs.princeton.edu/cass/papers/mplsh_vldb07.pdf.
15. T. S. Teixeira, G. Teodoro, E. Valle, and J. H. Saltz, "Scalable locality-sensitive hashing for similarity search in high-dimensional, large-scale multimedia datasets," arXiv:1310.4136 (2013); https://arxiv.org/pdf/1310.4136.pdf.
16. Z. Yang, W. T. Ooi, and Q. Sun, "Hierarchical, non-uniform locality sensitive hashing and its application to video identification," in *Proceedigns of the IEEE International Conference on Multimedia and Expo ICME,* IEEE Cat. No. 04TH8763 (2004), Vol. 1, pp. 743–746. https://www.comp.nus.edu.sg/ ooiwt/papers/lsh-icme04-final.pdf. Accessed 2018.
17. V. Stegailov, N. Orekhov, and G. Smirnov, "HPC hardware efficiency for quantum and classical molecular dynamics," in *Proceedigns of the International Conference on Parallel Computing Technologies* (Springer, 2015).
18. G. Smirnov and V. Stegailov, "Efficiency of classical molecular dynamics algorithms on supercomputers," Math. Models Comput. Simul. **8**, 734–743 (2016).
19. M. Armbrust, R. Xin, C. Lian, Y. Huai, D. Liu, J. Bradley, and M. Zaharia, "Spark sql: Relational data processing in spark," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, 2015,* pp. 1383–1394. https://amplab.cs.berkeley.edu/wp-content/uploads/2015/03/SparkSQLSigmod2015.pdf. Accessed 2018.
20. A. Agarkov, T. Ismagilov, D. Makagon, A. Semenov, and A. Simonov, "Performance evaluation of the Angara interconnect," in *Proceedings of the International Conference Russian Supercomputing Days, 2016,* pp. 626–639. https://www.dislab.org/docs/rsd2016-angara-bench.pdf. Accessed 2018.
21. P. Erdős and A. Rényi, "On random graphs," Publ. Math. Debrecen **6**, 290–297 (1959). http://snap.stanford.edu/class/cs224w-readings/erdos59random.pdf. Accessed 2018.