2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom)

Analysing and Predicting the runtime of Social Graphs

Rana Maher*, David Malone†
Hamilton Institute
Maynooth University, Kildare, Ireland
Email: *rana.maher.2016@mumail.ie, †david.malone@nuim.ie

***Abstract*—The explosion of Social Network Analysis (SNA) in many different areas and the growing need for powerful data analysis has emphasized the importance of in-memory big data processing in computer systems. Particularly, large-scale graphs are gaining much more attention due to their wide range of application. This rise, accompanied by a massive number of vertices and edges, led computations to become increasingly expensive and time consuming. That is why there is a move towards distributed systems or Big Data cluster(s) to provide the required computational power and memory to handle such demand of huge graphs. Thus, figuring out whether a new social graph dataset can be processed successfully on a personal machine or there is a need for a distributed system or big-memory machine is still a remaining open question. In this paper, we try to address this question by providing a comparative analysis for the performance of two of the most well known SNA tools for performing commonly used graph algorithms such as counting Triads, calculating Degree Distribution and finding Clusters which can give an indication of the possibility of carrying out the work on a personal machine. Based on these measurements, we train different supervised machine learning models for predicting the execution time of these algorithms. We compare the accuracy of the different machine learning models and provided the details of the most accurate model that can be exploited by end users to better estimate the execution time expected for processing new social graphs on a personal machine.**

***Index Terms*—Social Graphs; Graph Analytics; Social Network Analysis; Graph Algorithms; Performance; Predictive Modeling**

I. Introduction

Social graphs are popular structures for modeling relationships, interactions and communication between users, organizations or even groups. They are the most commonly used way of modeling the shared information and ideas over the Internet. Facebook, Twitter and LinkedIn are the most popular social networks nowadays. The way we deal with these networks is through social graphs that demonstrate the nodes/people and the edges/relationships in the social network. Network analysis is a collection of techniques to calculate various metrics for a social graph. These measurements can illustrate the topology of the network, define how people are connected to each other and who are the top influential people. They also provide information about communities, roles and user actions. In recent years, there has been an observed increase in the number of large online social networks, many of them have a massive number of users that can reach hundreds of millions of users [1]. Analysing these social network databases can provide rich information which can be beneficial for a wide range of applications and areas but is sometimes considered an expensive and time consuming. This is due to the expected super-linear increase in the computational time and therefore speed and scalability should be key challenges of social network analysis.

In order to handle huge real-world network analysis problems, distributed clusters may be required to accommodate "real-world" graph sizes. Alternatively, big-memory machines that can do a highly interactive analysis that can have advantages over distributed clusters [2]. A long computational time may be needed to handle any graph analytics like community detection, node ranking, computing shortest paths, number of triads, degree distribution and connected components. The need to have an efficient computational tool/model or even a query language to use with this social graphs has been addressed by many researchers e.g. [1], [3], [4].

What if we have an option to run our analytics on different computational platforms? How could we predict which platform is most suitable? this will be the goal of our work in this paper, to be able to predict the execution time of graph algorithms for unseen graphs using two of the most commonly social network analysis tools, but to reach this predictive execution time we will need first to know how currently available tools perform on a personal machine. There are many social network tools and libraries that can perform a set of operations, features and various algorithms with many functionalities for graph analysis of this rich data and information within the graphs. For example, SNAP[1], Gephi[2], NetwrokX[3] and Gremlin[4] are some of the most commonly used tools in the community.

The main contribution of this paper is a performance comparison study between social network analysis tools that can enable the prediction of execution time for unseen social graphs. We introduce an overview for the related work in Section 2. Various features for SNA software based tools (e.g., SNAP) and Query based tools (e.g., Gremlin) are described in Section 3. In Section 4, we show our comparative performance analysis between SNAP and Gremlin by running a set of

[1] http://snap.stanford.edu/snap
[2] https://gephi.org
[3] https://networkx.github.io
[4] https://github.com/tinkerpop/gremlin/wiki

978-1-5090-3936-4/16 $31.00 © 2016 IEEE
DOI 10.1109/BDCloud-SocialCom-SustainCom.2016.62

IEEE computer society

Authorized licensed use limited to: Maynooth University Library. Downloaded on February 09,2022 at 10:47:18 UTC from IEEE Xplore. Restrictions apply.

popular and commonly used algorithms on different real world social network sizes. In Section 5, we show the results of training and comparing various machine learning models with detailing the out-performed model. Section 6 concludes the paper and highlights some potential future work.

II. Related Work

Perez et al. in [2] proposed Ringo, a big-memory graph analytics tool, which supports interactive graph analytics of millions of edges through merging big-memory machines that can outperform all other distributed systems. The authors showed that a single machine with big-memory can provide an efficient platform for doing graph analytics. Distributed graph systems like Pregel that support parallel graph algorithms on multiple machines and support adoption of a Bulk Synchronous Parallel (BSP) model was proposed by Malewicz et al. in [5] and also GraphLab in [6], a distributed system for data mining and machine learning. In [7] Kyrola et al. presented the performance of GraphChi, a disk-based system on a PC that supports evolving graphs overtime, GraphChi has a low memory requirements which designed specially for computation on big-scale graphs. The authors performed a comparison between GraphChi and other distributed systems like Spark [8], Hadoop [9], PowerGraph [10] and GraphLab, it was found that PowerGraph can compute graph analytics using large cluster much more faster than using GraphChi on just a single machine. The comparison was performed on PageRank, one of the most popular graph algorithms. It has been shown that GraphChi can provide high performance for different purposes.

Seo et al. in [11] claimed that the performance of Datalog, a declarative logic programming language that is usually used as a query language [12], is not competitive with other low-level languages in the past. However, it allows the expression of many graph algorithms and supports recursion and high-level semantics which consequently allow optimization in time and parallelization. A high level query language for graph analytics named SociaLite was proposed by the authors as a Datalog extensions for powerful analysis on graphs. The authors performed a comparison for the execution times for running a shortest path graph algorithm on different benchmarks like Giraph [13] and Hadoop and then compared their execution time with SociaLite concluding that the latter outperforms. The authors presented a comparison for the execution time in [1] between Datalog engines like Overlog [14], IRIS [15] and LogicBlox [16] for running shortest path algorithm on single machine, showing a better performance for LogicBox. However when compared with SociaLite, the latter showed a better performance than LogicBlox. Also, the authors proposed a comparison of SociaLite with other implementations by java almost around 50%.

From the above state of the art, we have found that tackling the performance issue to predict an estimated time needed for analysing graphs is a new area that can be fruitful for detecting the execution time of evolving graphs. To the best of our knowledge, this is the first comparative analysis that aims to find an estimate prediction for the execution time of graph analytics based on different benchmarks using a PC.

III. Investigated Network Analysis Tools

The increase of Social Network Analysis is driven by the rise of online networks specially human networks [17]. This rise has driven many researchers and developers to create and develop different approaches, algorithms and tools to easily apply graph mining and analytics. This led to having a plentiful number of publicly available frameworks that have many algorithms supporting the study and manipulation of data for any type of network. Our main concern will be how to select the right tool for the observed large-scale evolving graphs and decide which tool can suit your system design, graph size or even the algorithms that are to be used.

Our experiments will target a comparison between two types graph analytics tools. We have chosen to conduct this comparison between Query-Language-Based tools like Gremlin and Software-Based tools like SNAP. One of the reasons for targeting a Query Language tool and comparing it with a Software tool is that the query languages are usually for generic purposes and can enable many users to do social network queries in an easy and professional way without having a software background [1]. While, the reason for targeting SNAP in the comparison that it is C++ and python-based, so it is supposed to have a better computational time compared to other software tools that were mentioned above [2]. We concisely summarize the features of both tools as below:

A. Query Based Tools

We took Gremlin as our example of a query-based tool.

- **Gremlin**: A domain specific language for working with graphs, a graph based programming language developed for multi-relational graphs, named property graphs. The following are the main features of Gremlin:
  1) Supports complex graph traversals.
  2) Works over different frameworks, graph databases and graph processors.
  3) Used within the Java language as a virtual machine that has a direct access to Java based application.
  4) Combines between query language, network analysis and manipulation of graphs.
  5) Enables a wide range of users who do not have a software background to do efficient and easy queries.

B. Software Based Tools

We selected SNAP as a typical software-based tool.

- **SNAP**: A free general purpose network analysis and graph mining based package tool with the following features:
  1) It is written in C++.
  2) Provides a python interface (snap.py) [18] for use with python and runs on windows, Mac OS and Linux.
  3) Scales to huge networks with hundreds of million of nodes and edges.

Authorized licensed use limited to: Maynooth University Library. Downloaded on February 09,2022 at 10:47:18 UTC from IEEE Xplore. Restrictions apply.

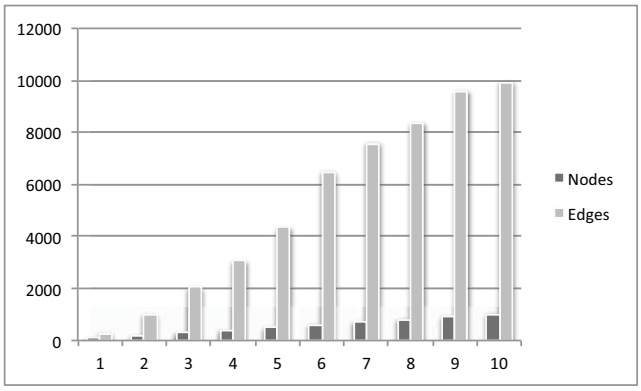


Fig. 1: Subgraphing Facebook Dataset.

4) Calculates the graph's structural properties, provide standard graph algorithms and different network structure measures.

IV. Experiments and Performance Evaluation

In this section, we start by outlining our testing setup and environment. Then we give an overview about the dataset used and our terminology in preparing and preprocessing the data for the experiments. We provide a comparative analysis for the results based on the SNAP and Gremlin tools and how their computational times differ for the same graph metrics.

A. Experiment Setup

- **Test Machine**: Most of the experiments were done on an Apple Mac Pro computer, 2.4 GHz Intel core 2 Duo, 3 MB cache size and 8 GB of main memory.
- **Test Protocol**: Each test is performed under the same setup and configuration. In the tests, we used multiple iterations and the mean execution time was reported. The datasets were held in-memory for each test. All our experiments were based on undirected graphs. It is worth noticing that the computational timing measured for both tools does not include the time taken to load the file into memory or writing the results into a file.

B. Dataset

The dataset considered in this paper is a public available Facebook dataset collected by Stanford University [19], a freely available real world graph dataset. This dataset represents a list of friends from Facebook, it presents a political affiliation social graph between members. The data was anonymized be replacing the internal ids for each user by new value. The circles consist of 4039 vertices and 88234 edges. Each vertex represents a user and an edge exists if any two users have same political affiliations. To perform our experiments, we divided this data into subgraphs to compare the results on different sizes of graphs with growing number of nodes and edges. We will explain the sub-graphing methodology in "Section IV-D" below.

C. Graph Analytics

We tested the execution time of three popular graph analysis algorithms for both tools:

- **Triads Count**: The aim of this algorithm is to count all the triangles or in other words the cliques of size 3. Counting the triads can be beneficial for many graph algorithms because they can be used to view similarity between structure of graphs [20] and can also be useful in community detection [21]. For example, in community detection algorithms, triads count is used to find the degree of edge support [22]. Hence counting triads is considered as a graph metric that is the basis of other analysis algorithms.

- **Degree Distribution**: This is a simple measure to count the number of edges for each node in the graph, it is based on the concept of neighborhood to find the vertices that have the most direct links to other vertices. It gives important clues about the structure of the network.

- **Detecting Clusters**: The aim of this algorithm is to find communities (clusters) or know how many unique clusters and what is the distribution of vertices within each cluster. This can be used in different community detection algorithms [23].

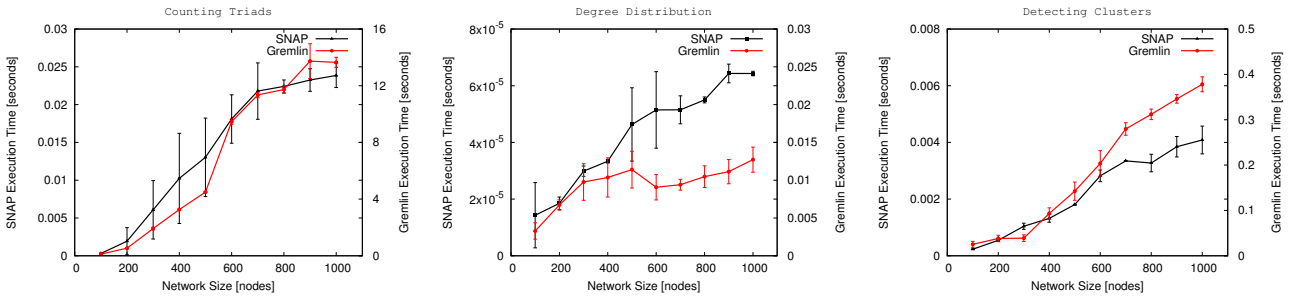
D. Subgraphs Preparation

To test the performance of the tools on different sized graphs, we extracted subgraphs of varying sizes from the complete graph. We divided the Facebook dataset into subgraphs, each subgraph was represented as a selected number of nodes and all their associated edges or links between them in the whole network. The selection of the nodes IDs is based on their ID value in the graph. For instance, with a subgraph of 100 nodes, we select the nodes with ID values between 0 and 100. We extracted 10 subgraphs using GetSubGraph function in snap.py library [18], this function takes the main graph and specified nodes IDs in vector form as their parameters and returns a subgraph induced by the nodes specified in this vector and the edges between these nodes. We repeated this process for having different subgraph sizes. The resulting subgraphs are 10 times smaller in edge count and nearly 4 times smaller in node count of the whole graph. Our strategy selected: $100, 200, 300, ...1000$ nodes and their associate edges so the network size varies from 100 to 1000 nodes and their edges numbers varies from 275 to 9890 edges as shown in Fig. 1. Hence, we evaluated the computational time on a graph growing constantly. Given this is a Facebook graph, the graph type is undirected with no multiple edges or self loops. These subgraphs were represented as an edge list in different files ready for analysis.

E. Performance Results

This subsection presents the performance results of the tools discussed in the previous section. The comparison is based on the execution time measured by both tools to calculate

Authorized licensed use limited to: Maynooth University Library. Downloaded on February 09,2022 at 10:47:18 UTC from IEEE Xplore. Restrictions apply.



(a) Counting Triads. (b) Degree Distribution. (c) Detecting Clusters.

Fig. 2: Comparative Analysis between SNAP and Gremlin for the execution time on different scales to measure the three metrics: Counting Triads, Degree Distribution and Detecting Clusters.

different metrics on different graph sizes. We expect that the computational time will be affected by the structure and the size of the graph. The elapsed time for running the algorithms will differ based on wether these algorithms access the edge list one or more times.

Referring to Fig. 2, we measured the execution time taken by each tool to measure each of Triad Count, Degree Distribution and Detecting Clusters on different sizes of subgraphs. In the first test, for the *Triads Count* as shown in Fig. 2a, we found a major difference in the performance of both tools. SNAP performed much better than Gremlin, the execution time taken by Gremlin was nearly 13 seconds to traverse 1000 nodes and 9800 edges while in SNAP it took nearly around 0.025 seconds and this is because the query used in Gremlin for counting the triads is likely touches and traverses every vertex in the graph to check their connection with their neighborhood and then check that their neighbors are connected so this led to traversing many vertices more than once. Counting triads of large-scale graphs usually require a fast algorithm, specially for graphs having billions of nodes and edges and it is preferable to be in a parallelized processes. For the second test, the *Degree Distribution* as shown in Fig. 2b, both tools performed better compared to calculating the triads. Unexpectedly, we observed that initially the execution time of Gremlin was high for the small subgraphs then it started to decrease when approaching a graph size of 500 nodes. It is worth emphasizing that we repeated the same experiment multiple times but while the dip is within the error bars, there does seem to be a trend. We do not have a clear justification of this behavior. The Gremlin query here traversed all the vertices to get their edge count. As for SNAP, it performed normally and as expected with an observed linear increase with the size of the network. In the last test for *Detecting Clusters* as shown in Fig. 2c, we observed that using SNAP took around 0.04 seconds for a graph with 1000 nodes while Gremlin took 0.4 seconds for the same graph so it is clear that SNAP is 10 times faster than Gremlin. The used Gremlin query in this test is based on the peer pressure vertex program algorithm, where every vertex assigned what is called by nominal value and if two vertices have same value therefore they are in same cluster and acquire the same cluster ID. Overall, SNAP performs much better than Gremlin in the three experiments. For the sake of increasing the accuracy of our reported results, we repeated each measurement for the execution time 20 times. The reason for the variance indicated by the error bars is seems to be due to runtime performance variability of the software and hardware. We repeated these tests with the same subgraph size by selecting different nodes to build the subgraph and found the execution time was similar. This indicates that the mean execution time is not affected much by the nodes selected and that the variation arises from runtime issues.

In the next section we apply machine learning techniques to create a predictive model for the execution time needed for each tool to process new unseen graphs.

V. Performance Prediction Modeling

With the continuos growth of data in various social graphs, learning how to take decision based on this data to improve business or provide solutions is almost an important need in different areas. Consequently learning a model for the performance issues can be useful for making decisions on where to run a graph analysis job. That is our main reason for applying machine learning algorithms to learn how to quickly perform graph analytics. We applied different regression machine learning models in order to select the best model that will predict the execution time of unseen graph based on two main features that are usually known in any social graph; nodes and edges. Our target is to present an approximate model as a computational technique that provides a relation between the execution time and the structure of the network.

A. Prediction Models Overview

Regarding the regression models, we trained using the results of the performance analysis on four popular regression models: Support Vector Machine (SVM), Multivariate Adaptive Regression Splines (MARS), M5 and Boosting.

- **SVM**: The SVM model is considered for both regression and classification problems based on detecting if data can be categorized or not and also based on the value of the

Authorized licensed use limited to: Maynooth University Library. Downloaded on February 09,2022 at 10:47:18 UTC from IEEE Xplore. Restrictions apply.

linear combination of the input features. SVM is capable to represent non-linear relationships in a linear fashion using a kernel function, which is a method for using a linear algorithm to solve non-linear problems [24].

- **MARS**: The MARS model usually uncovers important data patterns effectively, it is more flexible than other regression models and does not require any data preparation. The MARS model can be interpreted easily through the existence of the *hinge function* which partitions the input data automatically and works more appropriate for numeric variables make them work efficiently for numeric data. A pair of *hinge function* is usually written as $h(x-a)$ and $h(a-x)$ [24].
- **M5**: The M5 is a model tree algorithm. Model trees are used for the approximation and modeling complex non-linear problems. It is a promising model for prediction of numerical problems and it is popular by its robustness and efficiency [24].
- **Boosting**: The Boosting model is known to be a powerful predicting model. It is widely used in different applications. The algorithm of the boosting model gets initialized at first with the best guess (e.g., the mean value) and then the gradient is calculated, a model is then fit to minimize what is called a *loss function* and the current model is added to the previous one. This is repeated according to the number of iterations specified by the user [24].

B. Application to Performance Prediction

For the sake of training the machine learning models and achieving better accuracy, we extended our measurements to 50 various sizes of the same dataset and calculated the execution time for each of the three algorithms for both tools. We split our dataset formed from the 50 samples to a training dataset of 35 samples and 15 samples for the test dataset. We used the training set for estimating the coefficients of the different machine learning models whilst the test set was used for evaluating their performance. We used R to apply the machine learning models. The resulting performance profile appeared to have an observable difference between the four models in terms of Root Mean Square Error (RMSE).

C. Prediction Results

Referring to Fig. 3 and Fig. 4, the graphs represent the RMSE for each model for predicting the execution time for each of the Counting Triads, Degree Distribution and Detecting Clusters using SNAP and Gremlin as shown in the figures. It is clear that for both tools, the Boosting model had the highest RMSE for all of the three metrics. On the other hand, we found that the best model with the minimum RMSE for both tools regarding the three graph metrics is MARS. For SVM and M5, the first outperformed the latter for *Triads Count* and *Detecting Clusters* while on the other side M5 outperformed for calculating the *Degree Distribution* for both SNAP and Gremlin. Therefore, our proposed prediction will be based on MARS model since it showed the best performance for all metrics using both tools. Hence, we derived our MARS based model for predicting the execution time for measuring each metric illustrated by Eq. 1 where the coefficients (a, b, c, d, and e) along with the hinge functions (h_1, h_2, h_3, and h_4) for each metric for both tools are defined in Table I and Table II respectively. We denoted by ST, SD and SC as the Counting Triads, Degree Distribution and Clusters respectively for SNAP. Similarly, GT, GD and GC for same metrics but for Gremlin.

	a	b	c	d	e
ST	-1.5*10^{-4}	1.7*10^{-6}	-1.5*10^{-6}	-4.3*10^{-7}	3.8
SD	2.9*10^{-4}	-2.9*10^{-7}	2.1*10^{-7}	-7.5*10^{-8}	2.7*10^{-8}
SC	0.003	-0.0000004	0.0000002	0	0
GT	-1.05	0.009	-0.012	-0.002	0.002
GD	2.7	-6.7	1.4	0	0
GC	0.05	-0.00009	0.0005	-0.0009	0.00008

TABLE I: Special Coefficients for the hinge function of the MARS model for the execution time of ST, SD, SC and GT, GD, GC for SNAP and Gremlin respectively.

	$h_1(x)$	$h_2(x)$	$h_3(x)$	$h_4(x)$
ST	h(500-N)	h(N-500)	h(1522-E)	h(E-1522)
SD	h(420-N)	h(N-420)	h(1522-E)	h(E-1522)
SC	h(7153-E)	h(E-7153)	0	0
GT	h(500-N)	h(N-740)	h(1522-E)	h(E-1522)
GD	h(340-N)	h(N-340)	0	0
GC	h(340-N)	h(N-340)	h(N-820)	h(E-8620)

TABLE II: Hinge function of the coefficients a, b, c, d and e of the MARS model for the execution time of ST, SD, SC and GT, GD, GC for SNAP and Gremlin respectively.

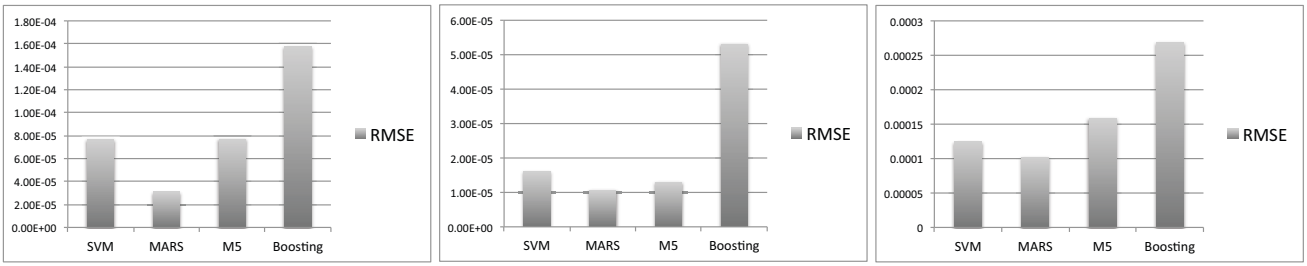
$$Time = a + b * h_1(x) + c * h_2(x) + d * h_3(x) + e * h_4(x)$$
$$where\ h_{1,2,3,4}(x) = \begin{cases} x & x > 0 \\ 0 & x \leq 0 \end{cases} \quad (1)$$

VI. Conclusion and Future Work

This paper proposed a performance comparative analysis between social network analysis tools using a personal machine. Our results related to two different types of tools: Software and Query based tools, the SNAP and Gremlin respectively. The Gremlin tool showed lower performance than SNAP, especially in calculating the number of triads in the graph. This suggests that using Gremlin should be accompanied by having a cluster to be able to parallelize many computations. On the other hand, SNAP performed efficiently on a personal machine and showed better results for different graph sizes for all the metrics, so it can be useful for anyone who is comfortable with python or C language. While SNAP can provide analytics on massive graphs it may lack features related to compatibility issues with graph databases and graph processors which are supported by Gremlin.

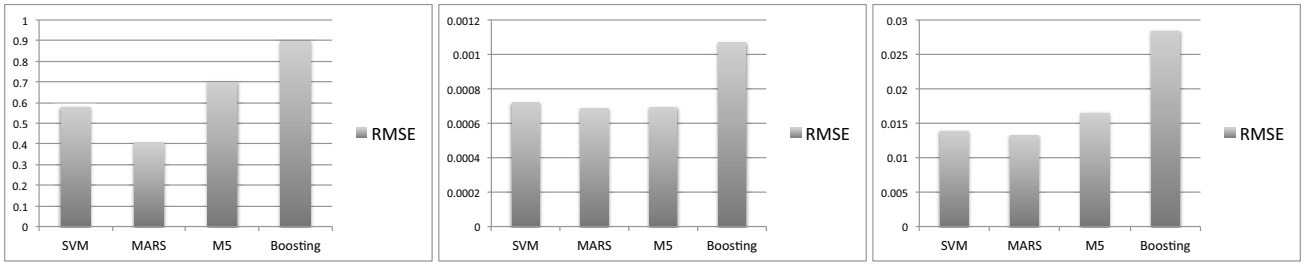
Next, in order to provide the end user with a prediction model for execution time, we trained different machine learning models to help take good decisions that facilitate timely analysis of the graphs. We tested four different models: SVM, MARS, M5 and Boosting on our test data which reports the

Authorized licensed use limited to: Maynooth University Library. Downloaded on February 09,2022 at 10:47:18 UTC from IEEE Xplore. Restrictions apply.



(a) Counting Triads. (b) Degree Distribution. (c) Detecting Clusters.

Fig. 3: RMSE for predicting the execution time (in seconds) for the the three metrics using SNAP.



(a) Counting Triads. (b) Degree Distribution. (c) Detecting Clusters.

Fig. 4: RMSE for predicting the execution time (in seconds) for the three metrics using Gremlin.

execution time for 50 subgraphs of different sizes for each of the three graph metrics for SNAP and Gremlin. We concluded from our experiments that MARS gives the best results and outperformed other models for both tools. Hence, we provided a mathematical formula based on MARS model for each graph metric for both tools to estimate/predict the execution time needed to analyze a given graph.

In the future, we will extend our work to account for the variance of the hardware used, such as available CPU resources and RAM size in big-memory machines (single big-memory machines) that can achieve faster random access required by graph algorithms and so are important in graph processing. Part of our ongoing work is to explore the execution time of distributed graph systems that are based on memory approaches like Pregel and GraphLab. As Gremlin showed a low performance on a personal machine and given that one of its advantages that it can be integrated with Hadoop (Gremlin/Hadoop) to allow parallel execution of Gremlin scripts as map reduce jobs on a Hadoop infrastructure, therefore we would like to evaluate it in a situation where it should perform better. Also, studying the impact on the computational time based on the structure of the network whether it is real world network or random network seems an interesting path to explore.

ACKNOWLEDGMENT

This publication emanated from research supported in part by a grant from Science Foundation Ireland (SFI) and co-funded under the European Regional Development Fund under Grant Number 13/RC/2077.

REFERENCES

[1] J. Seo, S. Guo, and M. S. Lam, "Socialite: Datalog extensions for efficient social network analysis," in *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*. IEEE, 2013, pp. 278–289.

[2] Y. Perez, R. Sosič, A. Banerjee, R. Puttagunta, M. Raison, P. Shah, and J. Leskovec, "Ringo: Interactive graph analytics on big-memory machines," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 2015, pp. 1105–1110.

[3] S. Elnikety and Y. He, "System support for managing large graphs in the cloud," in *Proceedings of the NSF Workshop on Social Networks and Mobility in the Cloud*. Citeseer, 2012.

[4] C. Yu, "Beyond simple parallelism: Challenges for scalable complex analysis over social data," in *Proceedings of the NSF Workshop on Social Networks and Mobility in the Cloud*, 2012.

[5] 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 *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM, 2010, pp. 135–146.

[6] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein, "Distributed graphlab: a framework for machine learning and data mining in the cloud," *Proceedings of the VLDB Endowment*, vol. 5, no. 8, pp. 716–727, 2012.

[7] A. Kyrola, G. Blelloch, and C. Guestrin, "Graphchi: Large-scale graph computation on just a pc," in *Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, 2012, pp. 31–46.

[8] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets." *HotCloud*, vol. 10, pp. 10–10, 2010.

[9] "Hadoop: An open-source framework for distributed processing of large datasets." http://hadoop.apache.org/.

[10] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "Powergraph: Distributed graph-parallel computation on natural graphs," in *Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, 2012, pp. 17–30.

[11] J. Seo, J. Park, J. Shin, and M. S. Lam, "Distributed socialite: a datalog-based language for large-scale graph analysis," *Proceedings of the VLDB Endowment*, vol. 6, no. 14, pp. 1906–1917, 2013.

Authorized licensed use limited to: Maynooth University Library. Downloaded on February 09,2022 at 10:47:18 UTC from IEEE Xplore. Restrictions apply.

[12] D. U. Jeffrey, "Principles of database and knowledge-base systems," 1989.
[13] "Giraph." http://giraph.apache.org/.
[14] T. Condie, D. Chu, J. M. Hellerstein, and P. Maniatis, "Evita raced: metacompilation for declarative networks," *Proceedings of the VLDB Endowment*, vol. 1, no. 1, pp. 1153–1165, 2008.
[15] "Iris: An open-source datalog engine." http://www.iris-reasoner.org/.
[16] "Logicblox." http://www.logicboxsoftware.com/.
[17] S. Wasserman and K. Faust, *Social network analysis: Methods and applications*. Cambridge university press, 1994, vol. 8.
[18] J. Leskovec and R. Sosič, "Snap.py: SNAP for Python, a general purpose network analysis and graph mining tool in Python," http://snap.stanford.edu/snappy, Jun. 2014.
[19] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection," http://snap.stanford.edu/data, Jun. 2014.
[20] J. W. Raymond, E. J. Gardiner, and P. Willett, "Rascal: Calculation of graph similarity using maximum common edge subgraphs," *The Computer Journal*, vol. 45, no. 6, pp. 631–644, 2002.
[21] G. Palla, I. Derényi, I. Farkas, and T. Vicsek, "Uncovering the overlapping community structure of complex networks in nature and society," *Nature*, vol. 435, no. 7043, pp. 814–818, 2005.
[22] J. W. Berry, B. Hendrickson, R. A. LaViolette, and C. A. Phillips, "Tolerating the community detection resolution limit with edge weighting," *Physical Review E*, vol. 83, no. 5, p. 056119, 2011.
[23] J. Leskovec, K. J. Lang, and M. Mahoney, "Empirical comparison of algorithms for network community detection," in *Proceedings of the 19th international conference on World wide web*. ACM, 2010, pp. 631–640.
[24] M. Kuhn and K. Johnson, *Applied predictive modeling*. Springer, 2013.

Authorized licensed use limited to: Maynooth University Library. Downloaded on February 09,2022 at 10:47:18 UTC from IEEE Xplore. Restrictions apply.