IMMM 2021

The Eleventh International Conference on Advances in Information Mining and Management

ISBN: 978-1-61208-864-8

May 30th – June 3rd, 2021

IMMM 2021 Editors

Guadalupe Ortiz, University of Cádiz, Spain

IMMM 2021

Foreword

The Eleventh International Conference on Advances in Information Mining and Management (IMMM 2021), held between May 30 – June 3rd, 2021, continued a series of academic and industrial events focusing on advances in all aspects related to information mining, management, and use.

The amount of information and its complexity makes it difficult for our society to take advantage of the distributed knowledge value. Knowledge, text, speech, picture, data, opinion, and other forms of information representation, as well as the large spectrum of different potential sources (sensors, bio, geographic, health, etc.) led to the development of special mining techniques, mechanisms support, applications and enabling tools. However, the variety of information semantics, the dynamic of information update and the rapid change in user needs are challenging aspects when gathering and analyzing information.

We take here the opportunity to warmly thank all the members of the IMMM 2021 Technical Program Committee, as well as the numerous reviewers. The creation of such a broad and high quality conference program would not have been possible without their involvement. We also kindly thank all the authors who dedicated much of their time and efforts to contribute to IMMM 2021. We truly believe that, thanks to all these efforts, the final conference program consisted of top quality contributions.

Also, this event could not have been a reality without the support of many individuals, organizations, and sponsors. We are grateful to the members of the IMMM 2021 organizing committee for their help in handling the logistics and for their work to make this professional meeting a success.

We hope that IMMM 2021 was a successful international forum for the exchange of ideas and results between academia and industry and for the promotion of progress in the field of information mining and management.

IMMM 2021 Chairs:

IMMM 2021 Steering Committee
（刘安安）An-An Liu**,** Tianjin University**,** China

IMMM 2021 Publicity Chair
Marta Botella-Campos, Universitat Politecnica de Valencia, Spain
Daniel Basterretxea, Universitat Politecnica de Valencia, Spain

IMMM 2021

COMMITTEE

IMMM 2021 Steering Committee

（刘安安）An-An Liu, Tianjin University, China

IMMM 2021 Publicity Chairs

Marta Botella-Campos, Universitat Politecnica de Valencia, Spain
Daniel Basterretxea, Universitat Politecnica de Valencia, Spain

IMMM 2021 Technical Program Committee

Akhlaq Ahmad, Umm Al Qura University, Saudi Arabia
Zaher Al Aghbari, University of Sharjah, UAE
Stelios Andreadis, Information Technologies Institute (ITI) | Centre of Research and Technology Hellas (CERTH), Greece
Kiran Kumar Bandeli, Walmart Inc., USA
Nadezda Chalupova, Mendel University in Brno, Czech Republic
Despoina Chatzakou, Centre for Research and Technology Hellas, Greece
Seongah Chin, Sungkyul University, South Korea
Tommy Dang, Texas Tech University, USA
Qin Ding, East Carolina University, USA
Ahlem Drif, Farhat Abass University, Sétif 1, Algeria
Hannes Fassold, JOANNEUM RESEARCH - DIGITAL, Graz, Austria
(David) Dagan Feng, The University of Sydney, Australia
Paolo Garza, Politecnico di Torino, Italy
Alessandro Giuliani, University of Cagliari, Italy
David Griol, University of Granada, Spain
Gabriel Henrique de Souza, Federal University of Juiz de Fora, Brazil
Tzung-Pei Hong, National University of Kaohsiung, Taiwan
Yin-Fu Huang, National Yunlin University of Science and Technology, Taiwan
Muhammad Nihal Hussain, University of Arkansas at Little Rock, USA
Liliana Ibeth Barbosa-Santillan, University of Guadalajara, Mexico
Young-Gab Kim, Sejong University, South Korea
Cristian Lai, ISOC - Information SOCiety | CRS4 - Center for Advanced Studies, Research and Development in Sardinia, Italy
Jean-Charles Lamirel, Université de Strasbourg Equipe SYNALP (ex. INRIA TALARIS) - LORIA - Nancy, France
Mariusz Łapczyński, Cracow University of Economics, Poland
Yuening Li, Texas A&M University, USA
Chih-Wei Lin, Fujian Agriculture and Forestry University, China
An-An Liu, Tianjin University, China
Flaminia Luccio, Università Ca' Foscari of Venice, Italy
Francesco Marcelloni, University of Pisa, Italy

Table of Contents

Predicting Software Quality from Development and Release Factors

Rishita Mullapudi
Computer and Information Science
Gannon University
Erie, PA
Email: mullapud002@gannon.edu

Tajmilur Rahman
Computer and Information Science
Gannon University
Erie, PA
Email: rahman007@gannon.edu

Joshua Nwokeji
Computer and Information Science
Gannon University
Erie, PA
Email: nwokeji001@gannon.edu

***Abstract*—Long lasting sustainable systems require quality software releases. If a new version of the software encounters relatively fewer post-release defects, i.e., bugs, then we can consider that version as a better quality release. In the competitive world of faster release and shorter release cycle based development, it is challenging to deliver a quality release of a software product. Predicting the release quality certainly helps developers to take precautions and measures to prevent post-release bugs. Although many researchers studied software quality prediction, a lack of robust empirical study on software development historical data to predict their impact on software release quality has been observed. In this study, we predict the release quality of Eclipse Equinox project by constructing a decision tree model from six factors, such as code changes (churns), commits, churns in test-files, churns in config-files, last-minute-change, etc., observed from the historical data extracted from the version control system. Such development and release factors will give us a better understanding on how the developers' activities affect the quality of a software release. Five quality levels, i.e., classes are used in our classification model from the Eclipse bugs depending on the presence of different levels of severity of bugs. Furthermore, we will construct three more models, Naïve Bayes, K-means Clustering, and Linear Regression, and will compare the accuracy of prediction. The outcome of this study will be a set of classification models built on the six development factors and an insightful comparison among them.**

***Keywords*—Software Quality, Release Quality, Software Quality Model, Open Source Software, Decision Trees**

I. Introduction

One of the objectives of software development is to achieve a high level of customer satisfaction [20]. In general, quality is defined as the ability of a product to satisfy the needs and expectations of customers. Software quality focuses on making the customer happy by providing a satisfactory outcome of the software application with an uninterrupted user-experience. Moreover, explicit attention to the quality factors may save the software life-cycle cost significantly [6]. Various approaches and frameworks [21] [22] [23] for measuring software quality have been proposed in literature. However, in this paper, we use post-release bugs to measure software quality.

Software quality has been measured in various techniques. Wehaibi et. al. examined the impact of self-admitted technical debt as a measure of software quality [10]. On the other hand, Araujo et. al. [1] used code-quality as a measure of software quality.. However, the majority of the studies have emphasized on predicting software quality issues to improve software quality [9].

The increasing popularity of rapid releases bringing the software products and new features into the market more frequently than before [3]. Maintaining the quality of the software product can be challenging in such a limited time-frame of release-cycles since testing in rapid release becomes challenging while manual system-integration test needs effective and efficient prioritization [11]. The effect of rapid releases on software quality also has been studied by Khomh et al. [24] for Mozilla Firefox. Since Eclipse is following a rapid-release model for their development, this increases our interest to choose the Eclipse Equinox project for this study.

A large amount of effort is involved in stabilization activities such as correcting coding standards, fixing bugs, adjusting configurations, twiking test files etc., during the testing or Quality Assurance (QA) period in a release cycle [7]. Although, in a rapid release, the effort during stabilization is not as large as the development effort, developers tend to rush towards the end of the development period right before the releasing phase starts [7]. Therefore, we are more interested to see whether the last minute changes have any impact on the post-release bugs, i.e., the overall software release quality.

To drive this research, we are interested in finding answers to the following research questions:

RQ1 How much code-change efforts are involved for a new release version?
Here, we quantify the number of commits and churns (code-changes) as a measure of effort to release a new version. We use a number of commits and churns as release factors to construct our prediction models.

RQ2 Do we see more post-release bugs when a new release version involves more test-files or configuration related files?
We quantify the code-changes in test files and configuration files in the commits to a release version. We use them as release factors to construct our prediction models.

RQ3 How significant are the last-minute changes to produce post-release bugs?
We consider the last one month window as the last-minute changes before publishing a release version. We want to see if we see more bugs where developers were more in a rush during the last one month of development.

TABLE I. COMMITS LABELED WITH BUILD AND RELEASE VERSION.

Commit	Build Version	Author Date	Release Version
bdfb311c27b7af506d9df031c0fa86c01bd2d88f	v200712031723	2007-11-29 14:21:48-05	3.1.2
850f068ac1f4264641adacc707c87f0f07a721de	v20090127-1212	2009-01-27 11:15:27-05	3.5.0
7c88166c83944184600862ecbe77935cbb4360ef	v200712031723	2007-11-29 14:55:38-05	3.1.2
2a34a6d8f11644b9ee80ac0cedbda0364f3f4116	v200712031723	2007-11-29 15:00:49-05	3.1.2
e539b0d4eeeb2686743eed41d2260a4bf6d92ef3	v200712031723	2007-11-29 14:27:35-05	3.1.2
adec7392cfe5ee6292a28d7520b567e897c8975b	v20071015	2007-10-15 18:20:27-04	3.4.0
86ad8d63826d6186aa433020f7d4b06330feec04	v20071015	2007-10-15 16:33:20-04	3.4.0

TABLE II. COMMITS DETAILS.

Commit	Churns	Old File	New File
bdfb3...2d88f	3	bundles/.../.../EclipseGeneratorApplication.java	bundles/.../.../EclipseGeneratorApplication.java
850f0...721de	2	bundles.../.../BuildPublisherAntTasks.launch	bundles.../.../PublisherAntTasks.launch
7c881...360ef	8	bundles/.../.../EclipseInstallGeneratorInfoProvider.java	bundles/.../.../EclipseInstallGeneratorInfoProvider.java5
2a34a...f4116	56	bundles/.../.../generator/Generator.java	bundles/.../.../generator/Generator.java

In this study, while finding answers to the RQ1 we obtain the numbers for various release factors from our data that will help us build our prediction models. Based on the results from our prediction models, we will be able answer RQ2 and RQ3. We define the quality levels of release versions depending on the presence of different severity levels of bugs. We use these quality levels as the classes of our prediction model for training and testing. Our main focus is to understand if there is any strong relationship between the quality levels and one or more of the release factors.

The following sections are organized as: Section II talks about the related studies in the literature and compares with our contribution in this paper. Section III explains the data source, data collection, and data pre-processing. Section IV explains our methodology, Section V explains our preliminary results obtained, statistics on the development/release factors that build our prediction models. Finally, in Section VI we summarize our research so far, explain how much progress we have made, and how much work still remaining.

II. LITERATURE REVIEW

Many researchers have predicted software quality using various prediction models. We are performing the study on Eclipse post-release bugs and six development and release factors. Similar to this, a study has been conducted by Misirli et. al. where they performed an explanatory analysis on eclipse beta-release bugs [18]. They considered six development related in-process metrics that have explanatory impact on beta-release bugs. The factors that they used are, age, number of edits, number of committers, average changed lines of code, last edit date and average time between edits. In our understanding a different set of factors may have a larger impact on the post-release bugs. Compared to their approach, we are considering a different set of development related factors (metrics) in each release version such as, number of commits, bug-fix commits, churn per file, churn per test-file, churn per config-file, and last-minute churns. Furthermore, we will investe each release version and predict the post-release bugs using other prediction models.

Zimmerman et. al. [9] predicted software quality from the historical data. Unlikely our approach, they considered bug-fix changes as a measure of software quality. They focused on analyzing the testing process to assess the impact on software quality in a rapid release model.

Seliya et. al. [17] used classification algorithms to predict software quality. They used C4.5 [17] and Random Forest decision-tree to build defect predictors. However, their focus was to investigate the cost-sensitivity of the learning mechanism on multiple data-sets collected from different software projects.

Araújo et. al. used four code quality features related to poor programming practice and evaluated the effectiveness of these features on post-release bugs in the procedural software applications [1].

Wehaibi et. al. [10] considered self-admitted technical debts as a measure of software quality. Phadke et. al. [5] considered fault-prone modules to measure and predict software quality. However, none of them used any classification tree to construct a prediction model.

Other prediction models have also been applied to predict software quality, such as the Bayesian network. A Bayesian network based approach has been taken to assess and predict software quality by Wagner et. al. [19]. They introduce the use of general quality models and show how the modelling of activities and facts in an organization helps define quality more precisely. They used the Bayesian network since it shows better performance for assessment and prediction incorporating variables with uncertainty.

We have not found any study which has followed the exact similar approach that we are following. Our approach is to construct a classification based prediction model based on the factors related to development and release from the historical development repository data. We would like to find if there is any strong relationship between post-release bugs and one or more of these factors. Furthermore, we will construct three more prediction models to find the best result and explain why such factors are significant to pay attention during the development activities in a release.

III. DATA

We used Eclipse Equinox [13] development historical data that we collected from their public Github repository which is a mirror of the official Eclipse repository [14]. We collected the the post-release bug reports from their Bugzilla portal [12].

A. Repository Data

First, we cloned the Eclipse-Equinox repository, which contains commits earliest from 2006 with a total of more than 6K commits. More than 63 developers contributed to this repository as of today. We then run a python script to extract the commit history and store them into a postgres database. The Python script extracts the commit data into 5 different tables, where all the commits are stored in the main commit table with author date (the date-time when the commit was made), and the commit message. Another table related to this table contains all the details about each commit such as lines modified, files modified/renamed, etc.

Another useful data we collect from the repository is the release-tags. Each time a build is created Github creates a tag with that build-commit and by extracting those tags from the repository we can track the release commits. Once we track the release commits, we then apply another python script to extract the git Directed Acyclic Graph (DAG) [15]. This script walks backward through the DAG traversing each and every commit all the way to the first one starting from a release (build) commit and labeling the commits on it's way with that release tag. This is how we know which commit belongs to which build version.

Once we have all the commits labeled with the build versions, we then look for what release versions the builds belong to. For this, we needed to put some manual effort to search for the build archives for various Eclipse-Equinox documentations [16]. We found lists of builds associated with release versions from various Eclipse documentations and online resources. Table I shows a segment of our commit data labeled with build versions and the release versions. We get the number of commits made to a release version from this table. Number of churns, test-files, config-files, churns in test files, churns in config files, these release factors we get from commit details. Table II shows a portion of our commit details data for the first four commits in table I. In this table, column ”Churn” contains the total number of lines of code changed (addition + remove), ”File” and ”New File” columns indicate if there is any file renamed, deleted, or added in that commit.

Table II shows a portion of our commit details data for the first four commits in table I. In this table, column ”Churn” contains the total number of lines of code changed (addition + remove). The “Old File” and “New File” columns indicate if there is any file rename, or deletion or addition in that commit. We get the release factors “total Churns”, “total files”, “churns in test file”, “churns in config files” from the table II.

We get the release factors “total Churns”, “total files”, “churns in test file”, “churns in config files” from the table II. To calculate the churn data we sum up the addition and deletion of lines of code in that commit. To calculate total churns in test files and config files, we first identify the test files and by search for the existence of the words “test” or “Test” in the file path. To identify the configuration files, we search for the existence of the words “conf” or “setting” but no “test” or “Test”. This is because there are test files to test configuration settings too and we want to consider those files as test files not configuration files.

TABLE III. ECLIPSE EQUINOX BUG DATA.

Bug ID	Release Version	Bug Severity
564065	4.17.0	Critical
566014	4.17.0	Normal
61632	3.0.0	Blocker
191487	3.0.0	Critical
67588	3.0.0	Major
285341	3.5.0	Normal

Last Minute Changes: Finally, another development factor we would like to investigate is how much changes developers are doing during the last one month of the release time-line. We are calling the last one month of changes as the “Last Minute Churn”.

B. Bug data

Eclipse-Equinox bug reports are stored in their Bugzilla portal [12]. We downloaded all the bug ‘id's as a “csv” file from that website. We wrote another python script to fetch the details of each of the bugs using the bug id. This python script pulls out the detailed information about each bug that contains date, bug-status, bug description, release version the bug was created, release version the bug was fixed, severity of the bug, and many other useful information. We stored this information into the same Postgres database. Table III shows a segment of the bug data we collected from Eclipse Bugzilla archive.

Eclipse Equinox uses bugzilla to store their bugs. Bugzilla allows us to categorize bugs in different types: “Enhancement”, “Trivial”, “Minor”, “Normal”, “Major”, “Critical”, “Blocker” etc., depending on the severity of the bugs.

“Enhancement” types of bugs are not actually defects, they are the limitations of a feature which probably because of missing that part during the initial planning for the feature, or during the development. “Trivial” bugs are the ones that do not have much impact on the performance or user experience. “Minor” bugs are the ones that have some impact but we can live with it for some time. No one will complain, or no significant performance issues at this point. However, it is a defect and we need to address this. “Normal” bugs are the ones that we need to address and schedule an appropriate scope of fix. Users have complaints but they can at least manage their work. “Major” bugs are the high-priority defects that are causing problems to the users and we need to fix this as soon as possible. “Critical” bugs are the bugs that are causing serious problems to the system. System is mal-functioning, users are having bad experience and having difficulties to do their work using the relevant feature. “Blocker” has the highest degree of impact which is blocking the affected feature, users are completely unable to use the feature.

Post-release bugs are an obvious fenomena in a software life-cycle. However, the presence of different types of severity bugs in a release indicates the level of quality of the release

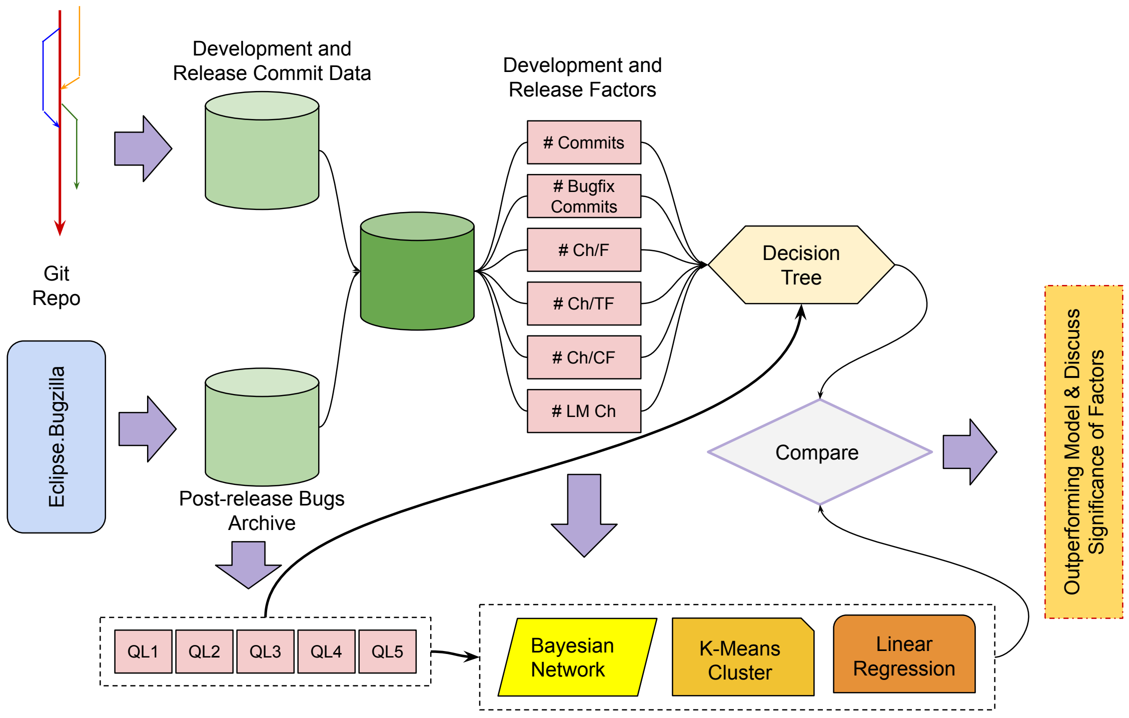

FIG. 1. THE METHOD OF OUR STUDY AT A GLANCE.

version. Our study focuses on predicting not just the number of post-release bugs, rather understanding the level of quality of the release based on the presence of different severity-levels of bugs.

IV. METHODOLOGY

Our primary prediction model is the decision tree using the six factors: “Churn per File” (Ch/F), “Total Number of Commits” (# C), “Bugfix Commits” (# BfC), “Churn per Test File” (# Ch/TF), “Churn per Config File’ (# Ch/CF)’, and “Last-minute Churns” (#LCh). To obtain the best performing model we will construct three more prediction models (Naïve Bayes, K-means Clustering, and Linear Regression) and compare the performance. Finally, we will discuss the impact of each of the factors on the results. Figure 1 presents our research-method at a glance.

We define the quality levels based on the following formula that considers high-impact bugs (hb), minor bugs (mb), major bugs (Mb), and total bugs (Tb). According to this formula, the magnitude (M) of a release is the product of the high-impact bugs (hb) and the ratio of minor and major bugs associated with a release version.

$$magnitudeM = hb \epsilon hb > 0 : hb * (mb + Mb) * 100/Tb \quad (1)$$

Magnitude of a release indicates how large is the impact of the bugs in that release. To measure that, we consider the percentage of major and minor bugs. The magnitude of the release is dominated by the presence of the high-impact bugs. Here, the number of high-impact bugs is the summation of critical and blocking bugs. We multiply the percentage of major and minor bugs by high-impact bugs to calculate the magnitude of a release. For example, the quality magnitude of release version 3.4.0 has been calculated like below:

$$m3.4 = 19 * (26 * 100/644) = 76.7 \quad (2)$$

Here, the number of high-impact bugs is 19, and we multiply by this only when this is $>$ 0. Table VI shows the magnitudes of release version 3.4.0, 3.5.0, and 3.6.0.

TABLE IV. QUALITY LEVELS (CLASSIFICATIONS).

Class	Quality Magnitude
QL1	0 - 50
QL2	51 - 100
QL3	101 - 150
QL4	151 - 200
QL5	201+

Release Magnitude: We define a threshold for the five quality levels based on the severity of bugs. The Severity levels include minor, major, critical, blocker. The five quality levels are represented as “QL1”, “QL2”, “QL3”, “QL4” and “QL5”. The thresholds for the quality levels are presented in Table IV. If a release magnitude falls within this range, we will label that release with the corresponding quality level. For example, the magnitude of release 3.4 is 76.7. Therefore, we can label this release with “QL2”.

Decision Tree: Our primary classification model is the supervised learning technique “Decision Tree (DT)”. Decision trees are easier to understand and categorize samples, and

TABLE V. Preliminary Stats on Development/Release Factors.

Version #	Ch/F	# C	# Ch/TF	# Ch/CF	# LCh	# BfC
3.4.0	33.54	433	109.16	10.90	40.02	-
4.2.0	14.53	59	13.38	12.25	22.66	-
4.5.0	10.67	72	10.04	3.40	42.5	-

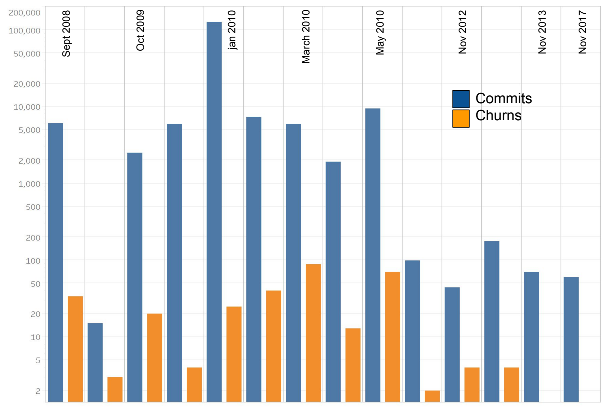

Fig. 2. Last-minute changes in version 3.6.0.

interpret the results. First, we prepare our model-data table with labeled data labeled using the quality levels. Decision trees do classification based on conditions at each level. In our case that condition will check the quantity of the six factors for each release, and calculate the magnitude using equation (1) to classify it to a quality level. We will train our DT with 70% of our labeled data and will test with 30% of the data.

V. Preliminary Results

Our data has been pre-processed and by this time we have obtained some preliminary statistics of the five of our six factors. Table V shows the preliminary counts for the five factors.

The column "# LCh" indicates the last minute changes in a release version which we collect from the last one month of churns in the release time-line as shown in figure 2. Eclipse Equinox stops making any commits in the repository three weeks before they announce their release version.

Figure 2 shows that as the developers in Eclipse approach towards the release, the ratio of churns per commit keeps increasing which has also been observed by Rahman et. al [7] in Linux and Google Chrome. This indicates that similar to Google Chrome and Linux, there is a little rush towards the release period observed in Eclipse Equinox as well.

We have defined the quality levels based on the release magnitudes from the bug data in Table IV. This magnitude will be used to determine different quality levels based on thresholds as explained in the methodology section.

TABLE VI. Quality Magnitudes of Releases.

Version #	Min	Maj	Crit	Block	Total Bugs	M
3.4.0	6	20	12	7	644	76.7
3.5.0	14	17	5	8	349	115.0
3.6.0	3	11	5	1	180	47.0

VI. Conclusion and Future Work

Software quality assurance is an important aspect in the software development lifecycle. It helps the software developers to measure the extents to which the product/software meets user's needs. The prediction of software quality has been studied in literature and various models have been proposed. However, achieving software quality still remains a major challenge to the software developers especially when the "Rapid Release" is in practice. The aim of this paper is to provide an approach and better understanding to support software quality improvement through prediction. We combine machine learning (ML) and artificial intelligence (AI) models to examine how code-changes and other relevant activities during development and release impact software quality measured through post-release bugs. Our study will provide an insight about developers activities and code changes to the developers which will improve existing methods where quality is usually assessed post-development. We believe that if we are able to identify a software is consuming high amount of effort in terms of commits or other quality factors at an early stage of the software development, then the software application is most likely to meet user expectations, satisfaction and thus have a higher quality.

We have not measured our sixth factor "bug-fix commits" yet. We need to apply Natural Language Processing (NLP) to understand the commit messages whether a commit is due to bug-fix or not. At this point, we will prepare our training and testing data. The classification model in this case would be the decision-tree which will fit the release to a quality-level based on the different threshold values of the quality-magnitudes as described in the methodology. Furthermore, we will construct three other prediction models: Naïve Bayes, K-means Clustering, and Linear Regression using the same development/release factors. We will compare the results, determine the best or outperforming model. We will also discuss the significance of each of the factors on the post-release bugs and will discuss the rationale behind. Furthermore, we plan to continue in this line of study. We plan to increase our sample size and expand the study to other open source software projects.

References

[1] C. W. Araújo, Z. Vanius, and N. Ingrid, "Using code quality features to predict bugs in procedural software systems." In Proceedings of the XXXII Brazilian Symposium on Software Engineering, pp. 122-131. 2018.

[2] R. Chopra, "Software quality assurance: a self-teaching introduction". Stylus Publishing, LLC, 2018.

[3] K. Beck and A. C. Andres. "Extreme programming explained: Embrace change. 2-nd edition." (2004).

[4] AT Misirli, B. Murphy, T. Zimmermann, and A. B. Bener, "An explanatory analysis on eclipse beta-release bugs through in-process metrics." In Proceedings of the 8th international workshop on Software quality, pp. 26-33. 2011.

[5] A. A. Phadke and E. B. Allen, "Predicting risky modules in open-source software for high-performance computing." In Proceedings of the second international workshop on Software engineering for high performance computing system applications, pp. 60-64. 2005.

[6] W. B. Boehm, J. R. Brown, and M. Lipow, ”Quantitative evaluation of software quality.” In Proceedings of the 2nd international conference on Software engineering, pp. 592-605. 1976.
[7] M. T. Rahman and P. C. Rigby, ”Release stabilization on linux and chrome.” IEEE Software 32, no. 2 (2015): pp. 81-88.
[8] M. V. Mäntylä, B. Adams, F. Khomh, E. Engström, and K. Petersen, ”On rapid releases and software testing: a case study and a semi-systematic literature review.” Empirical Software Engineering 20, no. 5 (2015): pp. 1384-1425.
[9] T. Zimmermann, N. Nagappan, and A. Zeller, ”Predicting bugs from history.” In Software evolution, pp. 69-88. Springer, Berlin, Heidelberg, 2008.
[10] S. Wehaibi, E. Shihab, and L. Guerrouj, ”Examining the impact of self-admitted technical debt on software quality.” In 2016 IEEE 23Rd international conference on software analysis, evolution, and reengineering (SANER), vol. 1, pp. 179-188. IEEE, 2016.
[11] H. Hemmati, Z. Fang, M. V. Mäntylä, and B. Adams, ”Prioritizing manual test cases in rapid release environments.” Software Testing, Verification and Reliability 27, no. 6 (2017): e1609.
[12] Eclipse bugs. url: https://bugs.eclipse.org/bugs/xmlrpc.cgi, Accessed: March 2021.
[13] Eclipse-Equinox Github repository. url: https://github.com/eclipse/rt.equinox.p2. Accessed: December 2020.
[14] Eclipse-Equinox official repository. url: git://git.eclipse.org/gitroot/equinox/rt.equinox.p2.git. Accessed: December 2020.
[15] Gitlab documentation on Git DAG. url: https://docs.gitlab.com/ee/ci/directed_acyclic_graph. Accessed: March 2021.
[16] Eclipse-Equinox build archive. url: https://archive.eclipse.org/equinox. Accessed: March 2021.
[17] N. Seliya and T. M. Khoshgoftaar, ”The use of decision trees for cost-sensitive classification: an empirical study in software quality prediction.” Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 1, no. 5 (2011): pp. 448-459.
[18] A. T. Misirli, B. Murphy, T. Zimmermann, and A. B. Bener, ”An explanatory analysis on eclipse beta-release bugs through in-process metrics.” In Proceedings of the 8th international workshop on Software quality, pp. 26-33. 2011.
[19] S. Wagner, ”A Bayesian network approach to assess and predict software quality using activity-based quality models.” Information and Software Technology 52, no. 11 (2010): pp. 1230-1241.
[20] T. Dey and A. Mockus, ”Deriving a usage-independent software quality metric.” Empirical Software Engineering 25, no. 2 (2020): pp. 1596-1641.
[21] I. Atoum, ”A novel framework for measuring software quality-in-use based on semantic similarity and sentiment analysis of software reviews.” Journal of King Saud University-Computer and Information Sciences 32, no. 1 (2020): pp. 113-125.
[22] E. Stephen and E. Mit, ”Framework for measuring the quality of software specification.” Journal of Telecommunication, Electronic and Computer Engineering (JTEC) 9, no. 2-10 (2017): pp. 79-84.
[23] H. Schnoor and W. Hasselbring, ”Poster: Toward Measuring Software Coupling via Weighted Dynamic Metrics.” In 2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion), pp. 342-343. IEEE, 2018.
[24] F. Khomh, T. Dhaliwal, Y. Zou, and B. Adams. ”Do faster releases improve software quality? an empirical case study of mozilla firefox.” In 2012 9th IEEE Working Conference on Mining Software Repositories (MSR), pp. 179-188. IEEE, 2012.
[25] Eclipse transition to smaller release cycles. “https://www.eclipse.org/lists/eclipse.org-planning-council/ msg02927.html”. Accessed: March 2021.

An Effective Approach for Genetic-Fuzzy Mining Using the Graphics Processing Unit

Chun-Hao Chen[1], Yu-Qi Huang[2] and Tzung-Pei Hong[2, 3]
[1]Department of Information and Finance Management, National Taipei University of Technology, Taipei, Taiwan
[2]Department of Computer Science and Information Engineering, National University of Kaohsiung, Kaohsiung, Taiwan
[3]Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung, Taiwan
Email: chchen@ntut.edu.tw, cream08111230@gmail.com, tphong@nuk.edu.tw

***Abstract*—Association analysis is an important technique for finding relationships among the given transactions. In real applications, since transactions may have quantitative values, the fuzzy-set theory was utilized for mining fuzzy association rules. To extract useful rules, the given membership functions were the critical factor. The genetic-fuzzy mining approaches were thus presented to obtain appropriate membership functions to mine fuzzy association rules. However, the evolution process was time-consuming. In this paper, we then propose an algorithm to reduce the processing time using the graphics processing unit (GPU), namely the GPU-based Genetic-Fuzzy Mining algorithm (GPU-GFM). It first collects the chromosomes from the population and the chromosomes generated by genetic operators. Then, chromosomes are sent to GPU to calculate the fitness values. As a result, a fitness value matrix is returned. At last, when reaching the termination condition, the best chromosome will be outputted for mining fuzzy association rules. Experiments were also conducted on simulation datasets to show the performance of the proposed approach.**

Keywords-Association rule; genetic algorithm; fuzzy set; fuzzy association rule; graphics processing unit.

I. Introduction

Data mining is commonly used to extract knowledge from the given datasets, and the Apriori algorithm is the well-known technique to be utilized for discovering relationships among the transactions [2]. An association rule is an expression of the relevance between items. For instance, $X \rightarrow Y$ is an association rule, where X and Y are itemsets. It means that when someone buys the items in the X, then the customer has a high probability of buying Y at the same time. For example, a customer who buys *milk* and *jam* will also buy *bread* could be found as an association rule and represented as $\{milk, jam\} \rightarrow \{bread\}$.

The abovementioned rule mining approach can only be used to mine binary association rules [2]. In other words, items in the transaction can only be considered as to buy or not to buy, which limits the content of data analysis. However, in real applications, the purchased quantity exists and should be taken into consideration in the mining process. Therefore, by using fuzzy sets, many algorithms have been proposed for mining fuzzy association rules [8][9][10][14]. The main concept of those mining algorithms is that the quantitative values are first transformed into fuzzy representations using the given membership functions. Then, the fuzzy representations are employed to discover fuzzy association rules. For example, Hong et al. proposed an approach for mining fuzzy association rules from quantitative data [8]. Ouyang et al. proposed an algorithm to mine direct weighted and indirect weighted fuzzy association rules [14].

In those fuzzy association rule mining algorithms, the membership functions are given in advance. Because the predefined membership functions may not be appropriate for all kinds of datasets to mine fuzzy association rules, and because to obtain appropriate membership functions is an optimization problem, the genetic-fuzzy mining algorithms have then been proposed to obtain the membership functions for mining fuzzy association rules using various evolutionary algorithms, chromosome representations, genetic operators as well as evaluation functions [1][4][5][6][7][13][16][17]. However, the main problem of the existing approaches is the evolution process is time-consuming.

With the prevalence of General-Purpose computing on Graphics Processing Units (GPGPU), in this paper, we propose a GPU-based Genetic-Fuzzy Mining algorithm (GPU-GFM) for handling the problem. It first generates the initial population randomly. Then, the population is sent to GPU to execute the Max-Min-Arithmetical (MMA) crossover operator. The offspring and the original chromosomes will return to CPU. After that, the mutation operator is performed. To calculate the fitness values of chromosomes, all chromosomes and transactions are sent to GPU to calculate the fuzzy values. As a result, the fuzzy value matrix is returned to the CPU. At last, the chromosomes and the fuzzy value matrix are again sent to the GPU for calculating the fitness values of chromosomes. A fitness value matrix is then returned to the CPU. When reaching the termination condition, the best chromosome is outputted for mining fuzzy association rules. Experiments were also conducted on simulation datasets with different parameter setting to show the efficiency and effectiveness of the proposed approach.

II. Related Work

In this section, the genetic-fuzzy mining algorithms are stated in Section II.A. The graphics processing unit based optimization approaches are described in Section II.B.

A. Genetic-Fuzzy Mining Algorithms

Hong et al. proposed an algorithm that consists of two phases for mining fuzzy association rules [6]. In the first phase, the genetic algorithm has been utilized to obtain the membership functions according to the number of large 1-itemsets and suitability of membership functions in a chromosome. In the second phase, the derived membership functions are used to discover rules. To reduce the time for the evolution process, Hong et al. took the divide-and-conquer strategy into consideration and proposed another algorithm for solving the genetic-fuzzy mining problem [7]. The main concept is that every item has its own genetic process to find membership functions. The obtained membership functions are gathered for mining fuzzy association rules. Because various criteria should be considered for the optimization process, Alhajj et al. proposed a multi-objective genetic algorithm for automated clustering to obtain fuzzy association rules [1]. Considering multiple minimum supports, Chen et al. then proposed an optimization algorithm for finding membership functions for items at a certain level. Then, the obtained membership functions are employed to extract multi-level fuzzy association rules [4]. In addition, the multi-objective genetic-fuzzy mining algorithm has been proposed for discovering multi-level fuzzy association rules [5]. Matthews et al. proposed an evolutionary-based approach for mining temporal fuzzy association rules for web usage data [8]. Palacios et al. proposed an algorithm, namely FARLAT-LQD, for obtaining both suitable membership functions and fuzzy association rule from imprecise transactions [15]. They first use the genetic algorithm to membership function based on 3-tuples linguistic representation model. Then, the frequent-pattern tree-based algorithm is employed to mine fuzzy association rules. Ting et al. proposed an enhanced genetic-fuzzy mining algorithm for membership functions and rule discovery [16]. The main advantage of the algorithm is that it used the structure-based representation, which considered the structures of membership functions for chromosome encoding.

B. *GPU-based Optimization Approaches*

With the popularity of computational intelligence nowadays, we often rely on computers to find the near optimization solution using metaheuristic algorithms. However, it usually needs a lot of time to obtain the result. After the general-purpose computing on the graphic processing unit was launched in 2011 by NVIDIA, the GPU parallel processing was employed to speed up the evolution process. For instance, Yousef et al. designed the genetic algorithm with GPU to solve the university course timetable problem [18]. Benaini et al. proposed an optimization algorithm with GPU to solve the vehicle routing problem because the path should be arranged in a short time. As a result, the proposed approach significantly reduced the time cost of obtaining the routing path [3]. Due to the government policies and the increase in environmental protection awareness in recent years, the energy-saving and efficient dynamic flexible flow shop scheduling has become a dynamic problem worthy of studying. To maintain the original efficiency, the principle of energy saving must be taken into consideration. In addition, scheduling problems will change with the different situations, so the time cost is a major issue. Luo et al. executed the GA method by GPU for parallel calculation to reduce significantly the time cost [8]. In the field of 3D printing, it often hopes that the loss of materials is as small as possible. Therefore, the support material needs to be calculated to find the closest or best solution. Li et al. used the GPU to handle the optimization problem to discover the schedule [12].

III. Proposed GPU-based Genetic-Fuzzy Mining Algorithm

In this section, the framework of the proposed GPU-based Genetic-Fuzzy Mining algorithm (GPU-GFM) is illustrated in Section III.A. The pseudo code of the GPU-GFM is stated in Section III.B. Components of the GPU-GFM are described in Section III.C.

A. *The Framework of the GPU-GFM*

The GPU-GFM framework is shown in Fig. 1.

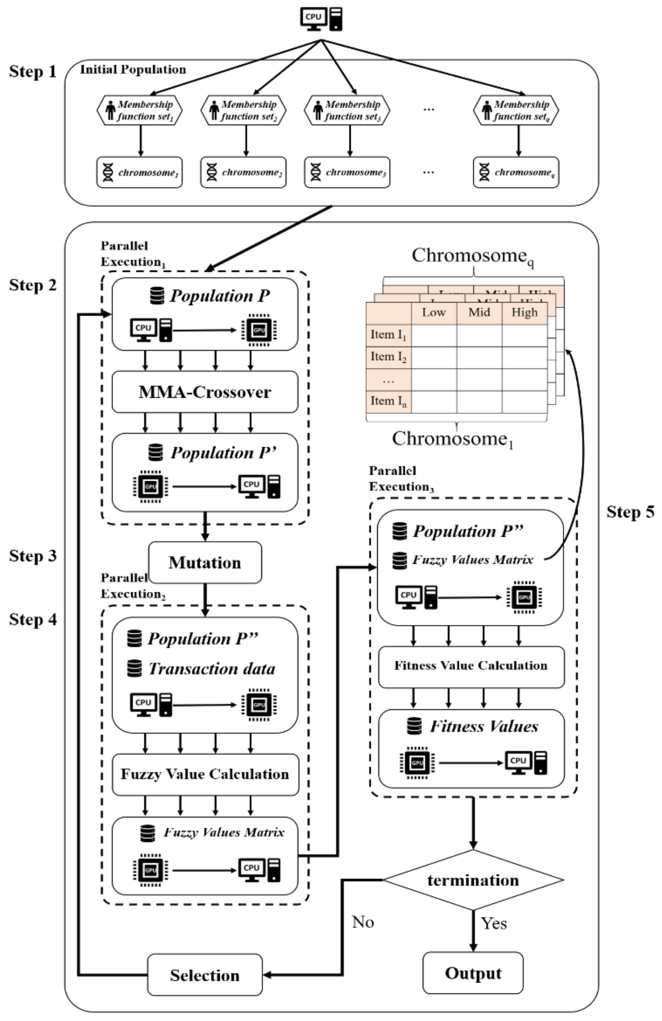

Figure 1. The framework of the GPU-GFM.

In Fig. 1, it shows that the proposed GPU-GFM contains five steps. They are: (1) The initial population *P* is generated randomly according to the predefined population size.; (2) The crossover operator is executed by GPU for speeding up the process to generate offspring which is merged to P to get P'; (3) The mutation operator is executed. After mutation, the population P'' is generated; (4) The GPU is utilized to transform quantitative transactions to fuzzy values for chromosomes; (5) Based on the fuzzy values matrix, the fitness values for chromosomes are calculated by GPU. Steps 1 to 5 will continue until reaching the termination condition.

In the following, we give a simple example to state the GPU-GFM. Assume that the population size is fifty. In Step 1, fifty chromosomes are generated randomly as the initial population P. Each chromosome represents a set of membership functions for all items.

In Step 2, assume that the crossover rate is 0.8. Forty chromosomes will be selected to generate offspring. Let two chromosomes as a pair. Thus, totally twenty pairs will be sent to GPU for offspring generation. The used crossover operator, the MMA crossover, will generate four candidate chromosomes for a given pair. Hence, after crossover, eighty offspring will be generated and sent back to the CPU. Then, the eighty chromosomes are merged to the P to form P'. In other words, P' has 130 chromosomes after the crossover operator.

In Step 3, for mutation operator, assume that the mutation rate is 0.04 and two chromosomes are mutated and added to P' to form P''. After mutation, the P'' has 132 chromosomes.

In Step 4, the quantitative transactions and P'' are sent to GPU for fuzzy value calculation. After calculation, a three-dimension matrix called the fuzzy value matrix will be generated. The index for the matrix including the chromosome number, item number, and fuzzy region number. Take (C_1, I_1, *Low*) is 5 as an example. It means the fuzzy value of the fuzzy region *Low* for item I_1 in chromosome C_1 is 5. The matrix is then sent back to the CPU for the next step.

In Step 5, the P'' and the *fuzzy value matrix* are sent to GPU again for calculating the fitness values of the 132 chromosomes. In the GPU, a thread is used to calculate the fitness value of a chromosome. It first calculates the number of large 1-itemset according to the given *fuzzy value matrix* and the predefined minimum support. Then, the suitability of the chromosome is calculated. After calculation, an array of fitness values is used to store the fitness value of chromosomes and returned to the CPU. At last, if the termination condition is reached, the best chromosome is outputted. Otherwise, it will go for the next generation.

B. Pseudo Code of the GPU-GFM

Based on the GPU-GFM framework, the pseudo code of the proposed algorithm is stated in Table I.

TABLE I. PSEUDO CODE OF GPU-GFM ALGORITHM.

Input:
Transaction data *TD.*
Parameters:
Population size *pSize*, crossover rate p_c, mutation rate p_m, generation *G,*
Population *P*, number of Items *itemNum*, minimum support *ms,*
Fuzzy Value Matrix *FVM.*
Output:
The best chromosome *BC.*
Procedure GPU-GFM:
1. *P* ← InitialPopulation(*pSize, itemNum*)
2. FOR *iteration* = 1 to ***G*** DO
3.   *GPU_P* ← cuda.memcpy_htod(*P*)
4.   *GPU_P'* ← MMA_Crossover(p_c, *GPU_P*, *GPU_ThreadIdx*)
5.   *P'* ← cuda.memcpy_dtoh(*GPU_P'*)
6.   *P''* ← Mutation(*p'*, p_m)
7.   (*GPU_P'', GPU_TD)* ← cuda.memcpy_htod(*p'', TD*)
8.   *GPU_FVM* ← FuzzyValueCalculation(*GPU_P'', GPU_TD, GPU_ThreadIdx*)
9.   *FVM* ← cuda.memcpy_dtoh(*GPU_FVM*)
10.   (*GPU_P'', GPU_FVM*) ← cuda.memcpy_htod(*P'', FVM*)
11.   *GPU_FitnessValues* ← FitnessValueCalculation(*GPU_P'', GPU_FVM, GPU_ThreadIdx*)
12.   *FitnessValues* ← cuda.memcpy_dtoh(*GPU_FitnessValues*)
13.   P ← selection(P'', *FitnessValues, pSize*)
14. END *iteration* FOR LOOP
15. BestChromosome ← selectBestChro(*P*, *FitnessValues*)

From Table I, the proposed algorithm first generates the initial population *P* randomly according to the predefined *pSize* (Line 1). Then, it starts the evolution process (Lines 2 to 14). The MMA crossover is then executed on GPU to generate offspring, and the results are stored in *GPU_P'* (Lines 3 to 4). The *GPU_P'* will return to CPU and store in P' (Line 5). The mutation operator is executed to get P'' (Line 6). To calculate fuzzy values of chromosomes, it sends P'' and transactions *TD* to GPU (Line 7). The fuzzy values of chromosomes are calculated (Line 8). The result *GPU_FVM* is returned to the CPU and stored in *FVM* (Line 9). For the fitness evaluation, the P'' and *FVM* are again sent to GPU (Line 10) for calculating fitness values (Line 11). The result *GPU_FitnessValues* is returned to CPU and stored in *FitnessValues* (Line 12). The selection process is executed to generate the next population (Line 13). Finally, if reaching the termination condition, the best chromosome is outputted (Line 15).

C. Components of the GPU-GFM

1) Encoding Scheme

In the proposed approach, a chromosome is used to represent a set of membership functions that are: $MFSet_1$, $MFSet_2$, …, $MFSet_i$, …, $MFSet_n$. The $MFSet_i$ means the membership functions for the i-th item. Let m linguistic terms are used for an item, then the $MFSet_i$ can be represented as ((c_1, w_1), (c_2, w_2), …, (c_j, w_j), …, (c_m, w_m)), where c_j and w_j are center and width of a membership function.

2) Initial Population and Genetic Operators

In the proposed GPU-GFM, the initial population is generated randomly. As to the genetic operators, the max-min-arithmetical (MMA) crossover operator and one-point mutation are employed to generate offspring. The elitist selection strategy is utilized for reproduction.

3) Fitness Evaluation

The fitness function used to evaluate a chromosome in the proposed approach is the same with the existing work [7]. The formula is stated as follows:

$$f(C_q) = |L_1| / \text{suitability}(C_q),$$

where $|L_1|$ is the number of large 1-itemsets that can be generated using the membership functions in C_q, and suitability(C_q) is used to avoid bad membership functions that are overlapping or separate too much.

IV. Experimental Results

In this section, experiments were made to show the performance of the proposed approach. The experimental environment is stated as follow: CPU: Intel(R) Core(TM) i5-9300 CPU @ 2.4GZ, GPU: NVIDIA GeForce GTX 1650. The proposed approach is implemented by Python 3.6.12. with the PyCUDA 2020.1 and CUDA v10.2 for deploying the algorithm on the GPU. The experimental datasets are generated by the IBM generator. By using the four parameters that are *T*: average transaction length， *I*: average maximum large itemset length， *N*: number of items, *D*: transaction size, different simulation datasets can be generated.

Experiments were first made to show the convergence of the proposed approach. After 1000 generations, the results are shown in Fig. 2.

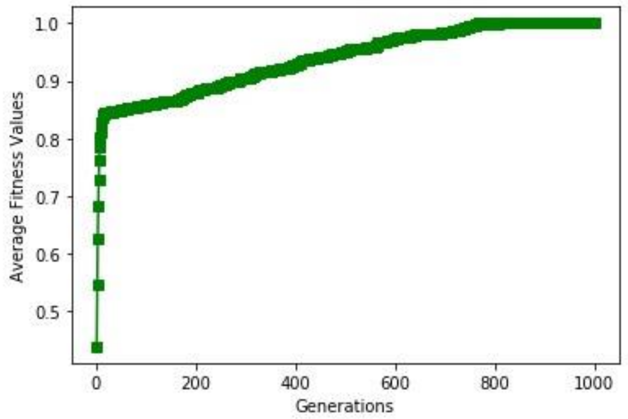

Figure 2. Convergence results of the proposed approach.

From Fig. 2, we can see that the average fitness values grow along with the increase of the generations, and finally converge to a certain value.

Experiments were then made to show the execution time of the proposed approach on the datasets with 170 items but different transaction sizes, including 10K, 30K, 50K, 90K. The results are shown in Fig. 3.

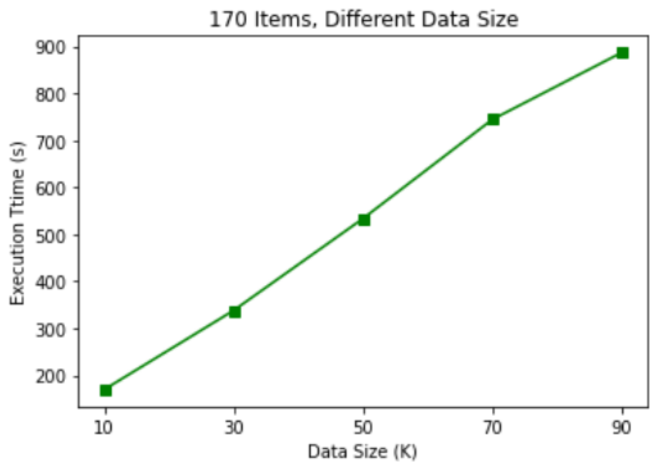

Figure 3. Execution time of the GPU-GFM on different transaction sizes.

From Fig. 3, we can observe that the execution time on different data sizes increase linearly. It indicates that the proposed approach is efficient. Then, the experiments on the datasets with 10K transactions but different numbers of items were made, and the results are shown in Table II.

TABLE II. Execution Time of the Proposed Approach with Different Number of Items .

Dataset	Execution Time (s)	Increasing Ratio
T2I2N0.032D10	85	-
T4I2N0.064D10	153	1.8 (= 153/85)
T6I2N0.096D10	217	1.4 (= 217/153)
T8I2N0.128D10	240	1.1 (= 240/217)
T10I2N0.16D10	298	1.2 (= 298/240)

Table II shows along with the increasing number of items from 32 to 160, the execution time increases from 89 to 298 seconds. From the increasing ratio, when we double the number of items from 32 to 64, the ratio is 1.8. The other three values are between 1.1 to 1.4. It means the execution time still increases linearly.

At last, comparisons of the proposed approach and the previous approach [6] in terms of execution time for a generation on the datasets with different transaction sizes are shown in Table III.

TABLE III. Comparisons of Proposed and Previous Approaches in Terms of Execution Time.

Data Size	Proposed Method	Previous Method	Speed-Up Ratio
10K	0.647 sec.	572.997 sec.	885
30 K	1.697 sec.	1710.271 sec.	1007
50 K	2.764 sec.	3154.801 sec.	1141
70 K	3.818 sec.	4537.069 sec.	1188
90 K	4.674 sec.	5172.524 sec.	1106

From Table III, we can easily observe that the proposed approach is better than the previous approach in terms of the execution time of the evolution process, and the highest speed-up ratio is up to 1188 times. From the experimental results, we can conclude that the proposed GPU-GFM is efficient significantly.

V. Conclusion and Future Work

Association rule mining is always an interesting research topic since it can be utilized to discover useful relationships among items. In real applications, transactions may have quantitative values. Fuzzy association-rule mining algorithms are employed to handle that. To extract more information, the genetic-fuzzy mining algorithms have then been presented to find membership functions automatically for fuzzy association-rule mining. Because the evolution process is time-consuming, in this paper, we thus propose an algorithm, namely the GPU-based Genetic-Fuzzy Mining algorithm (GPU-GFM), to speed up the evolution process. Experimental results show that: (1) the GPU-GFM is efficient no matter the increase of the number of transactions or items; (2) When compared to the previous approach, the highest speed-up ratio is up to 1188 times in terms of execution time. In the future, we will try to enhance the proposed approach to observe more useful rules, e.g., using all large itemsets instead of only large 1-itemsets as an evaluation function.

Acknowledgment

This research was supported by the Ministry of Science and Technology of the Republic of China under grant MOST 109-2622-E-027-032 and MOST 110AO12B.

References

[1] R. Alhajj and M. Kaya, "Multi-objective genetic algorithms based automated clustering for fuzzy association rules mining," *Journal of Intelligent Information Systems*, Vol. 31, No. 3, pp. 243-264, 2007.

[2] R. Agrawal, T. Imielinski and A. Swami, "Database mining: a performance perspective," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 5, No. 6, pp. 914-925, 1993.

[3] A. Benaini and A. Berrajaa, "Genetic algorithm for large dynamic vehicle routing problem on GPU," *International Conference on Logistics Operations Management*, pp. 1-9, 2018.

[4] C. H. Chen, T. P. Hong and Vincent S. Tseng, "Genetic-fuzzy mining with multiple minimum supports based on fuzzy clustering," *Soft Computing*, Vol. 15, No. 12, pp. 2319-2333, 2011.

[5] C. H. Chen, J. S. He and T. P. Hong, "MOGA-based fuzzy data mining with taxonomy," *Knowledge-Based Systems*, Vol. 54, pp. 53-65, 2013.

[6] T. P. Hong, C. H. Chen, Y. L. Wu and Y. C. Lee, "A GA-based fuzzy mining approach to achieve a trade-off between number of rules and suitability of membership functions," *Soft Computing*, Vol. 10, No. 11, pp. 1091-1101, 2006.

[7] T. P. Hong, C. H. Chen, Y. C. Lee and Y. L. Wu, "Genetic-fuzzy data mining with divide-and-conquer strategy," *IEEE Transactions on Evolutionary Computation*, Vol. 12, No. 2, pp. 252-265, 2008.

[8] T. P. Hong, C. S. Kuo and S. C. Chi, "Trade-off between computation time and number of rules for fuzzy mining from quantitative data," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, Vol. 9, No. 5, pp. 587-604, 2001.

[9] T. P. Hong, C. S. Kuo and S. C. Chi, "Mining association rules from quantitative data," *Intelligent Data Analysis*, Vol. 3, No. 5, pp. 363- 376, 1999.

[10] C. Kuok, A. Fu and M. Wong, "Mining fuzzy association rules in databases," *SIGMOD Record*, Vol. 27, No. 1, pp. 41-46, 1998.

[11] J. Luo, S. Fujimura, D. E. Baz and B. Plazolles, "GPU based parallel genetic algorithm for solving an energy efficient dynamic flexible flow shop scheduling problem," *Journal of Parallel and Distributed Computing*, Vol. 133, pp. 244-257, 2019.

[12] Z. Li er al., "A GPU based parallel genetic algorithm for the orientation optimization problem in 3D printing," *International Conference on Robotics and Automation*, pp. 2786-2792, 2019.

[13] S. G. Matthews, M. A. Gongora, A. A. Hopgood and S. Ahmadi, "Web usage mining with evolutionary extraction of temporal fuzzy association rules," *Knowledge-Based Systems*, Vol. 54, pp. 66-72, 2013.

[14] W. Ouyang and Q. Huang, "Mining direct and indirect weighted fuzzy association rules in large transaction databases," *International Conference on Fuzzy Systems and Knowledge Discovery*, pp. 128-132, 2009.

[15] A. M. Palacios, J. L. Palacios, L. Sánchez and J. Alcalá-Fdeza, "Genetic learning of the membership functions for mining fuzzy association rules from low quality data," *Information Sciences*, Vol. 295, No. 20, pp. 358-378, 2015.

[16] C. K. Ting, T. C. Wang, R. T. Liaw and T. P. Hong, "Genetic algorithm with a structure-based representation for genetic-fuzzy data mining," *Soft Computing*, Vol. 21, No. 11, pp. 2871–2882, 2016.

[17] T. C. Wang and R. T. Liaw, "Multifactorial genetic fuzzy data mining for building membership functions," *IEEE Congress on Evolutionary Computation*, pp. 1-8, 2020.

[18] A. H. Yousef et al., "A GPU based genetic algorithm solution for the timetabling problem," *International Conference on Computer Engineering & Systems*, pp. 103-109, 2016.