USING THE AVL ALGORITHM TO EVALUATE THE BUS FACTOR OF RESEARCH SOFTWARE PROJECTS

by

Ellie Kobak

A THESIS

Presented to the Department of Computer and Information Science

in partial fulfillment of the requirements for the degree of

Bachelor of Science

September 2022

THESIS ABSTRACT

Ellie Kobak

Bachelor of Science

Department of Computer and Information Sciences

August 2022

Title: USING THE AVL ALGORITHM TO EVALUATE THE BUS FACTOR RESEARCH SOFTWARE PROJECTS

This paper evaluates the accuracy of AVL Bus Factor (BF) estimations using commits through comparing the results to lines of code changes (LOCC) and the cosine difference of lines of code. In order to compare the BF values, the AVL BF algorithm for all three metrics was run on five open-sourced high performance computing projects. Three of the projects were then evaluated over a five-year span to determine any trends of BF across projects. The results across all five projects showed different values for the BF when using LOCC and cosine difference, with LOCC typically having a higher BF than commits and cosine difference. When comparing the three metrics on a year-to-year basis, the commits BF value was typically higher than the other two metrics. The implications of the commits having a higher BF estimation can lead to false sense of stability within projects, leading to the potential of more issues arising.

ACKNOWLEDMENTS

I would like to express my deepest thanks to Professor Boyana Norris for all her support and help throughout the entire thesis process. The project would not have been possible without her continuous guidance these past two years. I would also like to thank Professor Stephen Fickas for his assistance and advice. Additionally, I would like to acknowledge the help from the entire IDEAS group and the HPCL for their help and contribution to parts of this project. Lastly, I would like to thank all the faculty and my family and friends who have given their support throughout the process of this project.

TABLE OF CONTENTS

INTRODUCTION

The successful creation of large research software packages depends on effective collaboration among many developers with different backgrounds. Every developer works individually on their own computers and then uses version control software that enables every change to be tracked and saved enabling all developers on the project access to the most recent version of the project. Version control is such an essential part of the software engineering team development process that it used throughout computer science classrooms all the way too many tech companies creating their own niche version control software. The most commonly version control software used worldwide is called GitHub.

The project is saved on GitHub and referred to as repository or repo, which is accessible by every developer who has permissions to work on the project. The central repository is comprised of every file of code, document, or any other computer file needed for the project. Every developer can access the repo through any web browser or on their own computer, referred to as their local machine or working locally.

In order to prevent multiple people from making simultaneous changes on the same file, Git has a timeline tracker of changes called a branch. Figure 1 below depicts an example timeline of using version control. The central repo is often called the Main branch and is normally not directly accessible to developer changes. Instead, the developer clones, meaning they download a copy of the repo on their local machine. After they clone the project, the developer creates a new branch, where changes are made locally. The new branch is most commonly called a "feature" branch and is identical to the main branch when created. The difference between the feature and the main branch is that all the developer changes are only saved locally to the feature branch. Individual project changes are saved through "commits". For each commit, Git saves a

commit message written by the developer detailing the change, a timestamp, and a diff, which is all the information of every line of code changed since the last commit. A commit is only a local change. Therefore, when the developer wants everyone else on the project to view and have access to their updated code, they issue a pull request to merge their changes. A pull request details every commit and change between the feature branch and the main branch. Pull requests enable other developers to review each other's work and ensure correct functionality. Once the pull request is approved, the feature branch can be combined with a merge to the main branch, updating the central repository.

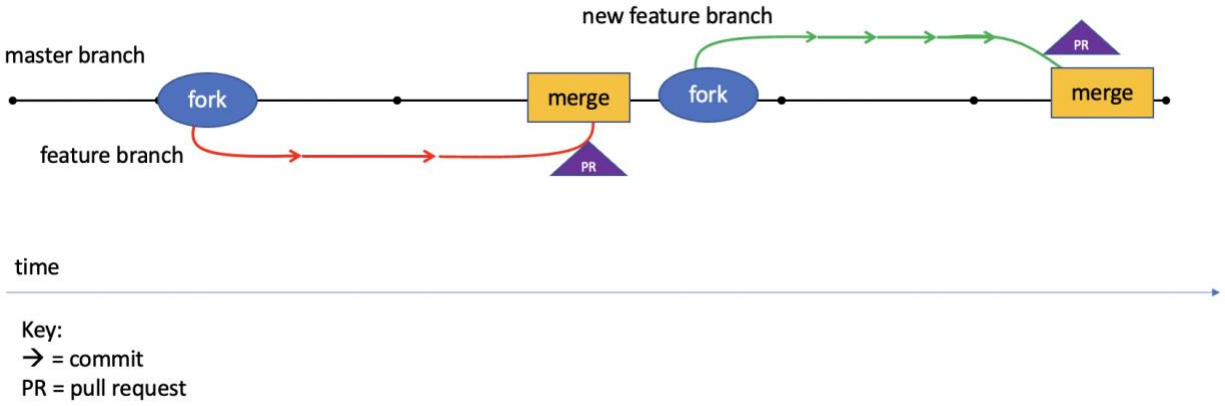

Figure 1: Image of Version Control Timeline

Not only is version control useful for allowing collaboration between developers, but the software is also an incredibly useful tool for project management. GitHub's tracking of every change in the project allows project stakeholders to closely monitor project development. Although Git is designed to assist the development in a team environment, the use of version control does not guarantee the success of a team. Every project will have setbacks and problems. Developers may employ several undesirable development practices, such as making too many

feature branches without merging, writing bad commit messages, or having badly designed commits, and in addition to poor software writing.

One method attempting to preemptively solve software development problems is through estimating the Bus Factor (BF or Truck Factor) for a project. The BF is the number of key developers who would need to be hit by a bus or leave the project, to send the project into such disarray, that the project would no longer be able to continue (Coplien and Harrison 2004). Natural language processing, which combines artificial intelligence, computer science, and linguistics, there have been attempts to analyze the Git data to calculate the BF and help prevent project issues. For example, if a project has a BF of five, meaning they absolutely require five developers in order to complete the project, and one developer goes on vacation midway through the project, the project will have to discontinue. However, if the manager knew the BF was five, they could have assigned a sixth or seventh developer to the project from the start. This situation is very common in the software development from too few developers being assigned to the project from the start or from developers leaving mid-project for any number of reasons. Therefore, if we can determine the most accurate BF for any given software project based on previous version control data, software developers and managers alike can use the BF to create more balanced project teams and help decrease future issues.

PROPOSED ARGUMENT

The proposed argument of this experiment is to determine if the AVL algorithm can more accurately determine the BF for a project based on the commits, lines of code, and the cosine of lines of code. The end goal of this experiment is to use the BF as a tool to reduce potential problems in software development.

RESEARCH QUESTION

There are two research questions that will be addressed.

1. Does the AVL algorithm more accurately determine the BF when comparing commits?
2. How does the accuracy of the BF differ when comparing number of commits, number of changed lines of code, and the cosine difference of the change of lines of code?

EXISTING LITERATURE

Attempting to estimate the Bus Factor is not new. Several others have tried to assess the Bus Factor of various projects using Git repositories. In 2011, Filippo Ricca et al. examined the difficulty of assessing BF using initial Truck Factor algorithms developed by Zazworka et al. in which each developer who had at least one commit was considered part of the project when determining the BF. In addition, the BF was examined using seemingly arbitrary percentages as thresholds to determine the importance of each developer, to which the authors acknowledge they could not trust the thresholds for they came from "a non-reliable source of information (a blog post)" (Ricca, Torchiano and Marchetto June 2011). The BF compared the approaches of Zazworka who used student projects as a preliminary BF algorithm with the methods used by blogger Siddharta Govindaray. Although they found similar results for smaller projects, Ricca et al. concludes the article admitting the methods tested do not account for large commits by one-person, similar changes in several files, or developers with multiple accounts, on top of several other limitations.

Further research attempts to improve the BF with degree of ownership and developer knowledge evaluations for files. Cosentino et al. used four methods to determine the degree of knowledge, last change of file takes all ownership, nonconsecutive changes (weighted and non-

weighted) and multiple changes considered equally (Cosentino, Cabot and Izquierdo March 2015). The degree of ownership helped determine the thresholds used for determining an essential developer when determining the BF. This paper simply developed a tool to calculate the BF but was implemented on a few private repositories. With this, there is no evaluation on larger open-source projects or discussion of the accuracy of their BF determinations.

Later research uses multiple algorithms to determine BF and then compares the results. Ferreira et al. use the AVL, RIG, and CST algorithms to compare the BF for the same open-source project. The RIG algorithm uses a blame-based approach where the blame is the developer who last modified a file. This algorithm found moderately successful results (Ferreira and Valente 2017). In the CST algorithm, the BF is determined by comparing the primary developer and a secondary developer. The algorithm determines the BF by the minimum number of primary developers.

In the third approach, the AVL algorithm relies on the degree of authorship (DOA), where a developer is considered the author if they committed more than 75% of the commits in a file. The algorithm runs on all the files in the project, incrementing the BF when an author is removed from each file. When 50% of the files from the project are removed the algorithm terminates. Using the AVL algorithm found the highest accuracy of results in determining the BF (Ferreira and Valente 2017).

Although Ferreira et al.'s study found success in determining the BF, the algorithms relied on number of commits or the developer who had the last commit in determining key developers for a project. These methods do not account for different coding styles of many small commits or few large commits. Due to the high degree of success of the AVL algorithm in determining BF, this study will use the AVL approach. Instead of running the algorithm's DoA

on the number of commits, this study will examine the DoA through the number of lines changed and on the degree of functionality of each line of code within a file. Therefore, we will have a more comprehensive understanding of the influence of each developer's work in a project and hopefully determine a more accurate BF.

METHODOLOGY

I attempted to find a better method to estimate the BF through examining previous work on computing the BF and combining it with existing computations done in the University of Oregon's High Performance Computing Lab (HPCL) for determining the lines of code, and the cosine difference of code changes. The Bus Factors were computed using Python3 within a Jupyter Notebook.[1]

Part 1: Data Collection

The data used in this project are from six open-sourced high performance computing projects. These projects have been inputted into a SQL database and have been organized and parsed for easier use for the IDEAs group in the HPCL. The database had the existing code enabling Git analysis. The projects are the following:

Project Name	Year Released	Total Contributors[2]	Description
Spack[3]	2014	1067	A package manager to run on multiplatform devices

[1] Link to Notebook: https://colab.research.google.com/gist/ekobak17/09a75f465a734a59626f6c400f5fcc3b/bf_computation.ipynb
[2] On github
[3] https://github.com/spack/spack

Ginkgo[4]	2017	22	A high-performance computing linear algebra library.
NWChem[5]	1994	53	Chemistry computations for high-performance computing.
Lammps[6]	2016	215	Simulator for large scale atomic and molecular massiveness.
Petsc[7]	1994	207	A toolkit for parallel computing for partial differential equations.
Hypre[8]	2004	42	A library of high-performance preconditions and solvers for parallel computing.

The last repo used to help develop and create the algorithms in this project was ideas-uo, which is a smaller repo that was used primarily for debugging and development.

Part 2: Algorithms

In order to calculate the BF, I first had to do research into how to define the bus factor. Due to the definition stating "number of key developers who would need to be hit by a bus or incapacitated, to send the project into such disarray, that the project would no longer be able to continue" (Coplien and Harrison 2004). This definition has been numerically defined differently in multiple studies, where in some studies, the authors even stated the numbers used in the metrics to determine the BF are arbitrary. For this reason, when reviewing past work, I paid

[4] https://github.com/ginkgo-project/ginkgo
[5] https://github.com/nwchemgit/nwchem
[6] https://github.com/lammps/lammps
[7] https://github.com/petsc/petsc, https://petsc.org/release/
[8] https://github.com/hypre-space/hypre

closer attention to the papers that had more reasonings behind how they chose and determined their metrics for determining the BF.

At last, I came across Ferreira et al.'s paper. In their paper, the BF was determined in multiple parts. First, they determined the Degree of Authorship (DoA), which is how influential each author is to the file.

The DoA for a file is defined as an author who has made an impact of 75% or more. Ferreira et al. based their experiment off the number of commits and therefore named a developer author of the file if the developer had made at least 75% or more of the commits for the file. They got this algorithm from the *Degree-of-Knowledge: Modeling a Developer's Knowledge of Code* by Fritz et al. where they accounted the DoA by who first created the file, subsequent changes, and accepting changes made by a different author (Fritz, et al. March 2014, 18). The article had no information regarding how they calculated the DoA based on the three factors above, nor did they include the accuracy of calculating the DoA with their algorithm.

Due to the high degree of unclarity based on if this algorithm worked, I decided the best way to determine the DoA was through my own metric. I modeled the algorithm off the description used in Ferreira et al. combined with the way the SQL data was stored. In addition, using a simpler algorithm to compute the DoA allows for more future scalability due to the use of table transformations (and dataframes which are easily mutable) instead of iterating over each row. This DoA allows different metrics and could easily be adapted to include specific time frames, allowing more concise BF computations. The pseudocode for determining the DoA is the following:

```
function DoA (metric, file, authors)
      DoA = 0
      metric_count = {}
```

```
total_metric = 0
for (all metric for file)
      if (metric by author in authors)
            metric_count {author: ++} // increase metric count
            total_metric++
max_author = max(metric_count)
DoA = max_author / total_metric
if (DoA >= .75)
      return [file, author]
return [file, NULL]
```

The actual code returns a Pandas dataframe consisting of the author's name and the file path. The metric parameter can be substituted for the lines of code, number of commits and cosine difference of code, or any other metric that is determined numerically. If a file does not have a clear author, the degree of authorship is considered null, and the file is not included when computing the BF. A dataframe is used because Pandas allows accessible and easily read information, especially for large data sets.

After the DoA is calculated for each file in the project, all the files that have a DoA greater than 0.75 are used as part of the BF computation. All the authors associated with these files are also stored and used in the computation. The next step of computing the BF relies on the AVL Algorithm.

The AVL Algorithm, which is named for the three inventors, Adelson, Velski & Landis, is a self-balancing binary search tree. An AVL tree must always have the height difference of the left and right sub trees no greater than 1, which is referred to the balance factor. An example of the self-balancing properties of an AVL tree with a root of 10 and a balance factor of 1 with a larger right subtree is followed. Let's say 5 wishes to be inserted into the tree. The addition of 5 increases the balance factor from 1 to 2. Due to the nature of the AVL tree needing to have a balance factor of one or less, the tree will undergo rotations, shifting the root to rebalance the

tree and maintain a legal balance factor. Let's say 12 wishes to be inserted into the tree instead of 5. Because 12 is greater than root, the 12 will go in the left subtree and will decrease the balance factor to 0 (assuming the left subtree is completely balanced at 0 before inserting 12). The AVL tree maintains the ability to rotate and self-balance when removing items from the tree as well. This ability to self-balance is crucial for the experiment.

The actual algorithm to compute the BF relies heavily on the AVL algorithm. The previously computed DoA, the author list, and metric are inputted into a function that calculates the BF. In the function, an author is removed one at a time. After an author is removed, all the files of which they are key developers are removed from the file count and the BF is incremented. When the file count reaches 50% of the total project files with valid authors, the algorithm stops running and the BF is returned. The pseudo code for this function is below.

```
function computeBF (num_files, authors):
      insert all metrics into AVL tree
      current_files = num_files
      bus_factor = 0
      while (current_files < (num_files // 2):
            remove author from authors
            if (author is author of file):
                  current_files -= files by author
                  remove metric from AVL tree
                  bus_factor += 1

  return bus_factor
```

In the actual code, the `authors` store the author and the numerical value of the associated metric in a dictionary. The use of the dictionary allows to keep track of which author is responsible for each file. When the metrics are inserted or removed from the AVL tree, they are done through the dictionary value of the author's name. The AVL tree is useful for the

algorithm because it stores all the values from the authors in a balanced manner, making the runtime of each removal O (log n).

An additional modification I made to the algorithm was to always remove the author with the minimum number of files they have DoA for first. This is because most projects have a couple of authors (or in some cases one author) who contributed significantly more than anyone else. Therefore, if the algorithm runs with random author removal, and there is one author with almost half of the file ownership, whenever they are removed the algorithm will terminate shortly after. Because a higher BF count shows more stability in a project, I decided to always remove the authors in an increasing order which will guarantee the same BF for each project every time it is run and also give a more conservative BF estimation.

Additionally, the importance of comparing the three metrics in this experiment is due to using a new DoA method. Most previous papers have computed the BF using commit data. Because I am changing the Ferreira et al. algorithm it is important to compute the BF of the commits, even though it will not be the most accurate in order to have some method of comparison between this experiment and other past work (Ferreira and Valente 2017 and Cosentino, Cabot and Izquierdo March 2015).

Part 3: Metrics Used

In Ferreira et al.'s study, they used the number of commits as the metric to evaluate the BF. Commits are not a great metric to use due to the vast coding styles. As illustrated in figure 1, with the two forks, a merge can consist of multiple different amounts of commits. One developer may write 100 lines of code per commit, and another may write 10. Commits can further differ with a developer committing after every change, and another member of the same team issuing one commit after finishing an entire file of work. Commits are an unreliable metric of work

through the examining the following file example. File A has five total commits and two authors. Author 1 wrote 120 lines of code and did one commit, Author 2 did four commits, one commit consisted of changing variable names, another commit adding a header, the third commit changing the format of spacing, and the fourth commit changing the variable names to the original name. Clearly Author 1 is the developer who did the most work on the file, but under Ferreira et al.'s method of measuring developer contribution, Author 2 would be assigned the key developer for file A. Thus, using commits is not a guaranteed measurement of developer knowledge for a project and to most accurately compute the BF.

Therefore, instead of using commits, in this experiment we used lines of code changes and the cosine difference of lines of code. Because each commit keeps track of every single line of code change, each file can be analyzed by how many lines each author contributed to the file. With the example of File A, using the lines of code to track author impact would make Author 1 the key developer for the file. However, using lines of code is not the most accurate measurement because Git counts removing spaces as changes in lines of code. For example, in File B, there are three authors, Author 1 likes to write code with lots of comments and in a second commit, goes back and removes unnecessary comments and spaces. Although they wrote 30 lines initially, after the second commit, the diff stated they had changed 50 lines of total code changes. Author 3 is an excellent programmer and writes very concise code and added 70 lines of code. Author 2 was then tasked in fixing the format to match company style and changed variable names and spacing, resulting in 30 lines of changes. Although Author 3 wrote the majority of the file's functionality, the BF algorithm would evaluate all three authors of having similar contributions. However, through using the cosine of lines of code changes, Author 1 and Author 2's changes will be viewed as less significant due to them changing characters in the

lines and not actually impacting the file. Therefore, cosine difference allows the most comprehensive understanding of how much contribution each author had based on the commit information.

These metrics are possible to use because of prior experiments in the IDEAS Lab evaluating the git data of open-sourced projects. The lines of code changed (LOCC) metric calculated through the commits detailing which lines have been added or subtracted. The cosine difference of the changes is calculated by a package that examines text differences through adding all the words in the changed lines to a word bank placing the added words in a column and deleted words in a row. The word differences then compare the columns and rows and assigns a numeric value of how much the line was changed. For this reason, a variable name change will classify as a very small change under the cosine metric but may be a larger change under LOCC. Using the cosine evaluation of lines of code changed allows a more detailed review of how each edit or author is changing the file over commits or LOCC.

Part 4: Evaluating Results

After computing the BF for multiple projects, I was able to compare the results. In a Jupyter Notebook, I used Pandas to generate visual representations to make sense of the data and compare the differences in BF between each metric. I computed the BF using the commits to serve as a baseline of accuracy for using the AVL algorithm approach to compare the two new metrics. For Hypre, Lammps, and NWChem, I ran the BF calculations on 2017 to 2021 to see how the BF differs year to year.

RESULTS

The results from running the AVL Bus Factor algorithm for all five projects are listed in the table 1.

AVL Algorithm Bus Factor Data		
Project Title	Metric	Bus Factor
Spack	commits	75
	locc	109
	cos	64
PETSc	commits	68
	locc	95
	cos	68
Hypre	commits	33
	locc	39
	cos	32
Lammps	commits	72
	locc	116
	cos	116
NWChem	commits	46
	locc	62
	cos	57

Table 1: AVL Algorithm Bus Factor Data

In most projects, the cosine, commits, and LOCC computation methods resulted in different Bus Factors. Further, in most projects, LOCC resulted in a higher BF than the other two metrics. In order to get a deeper understanding of the BF, I considered three projects over a five-year period to see how the BF changes. I chose the projects that had variations in sizes in order to compare a small, medium, and large project. I noticed that in most projects there were a small number of developers who had ownership of most files. Because the algorithm terminates after 50% of the files have been removed, I included a table for each of three projects containing the side-by-side comparison of maximum developer contribution to the BF in the Appendix. In most cases, the AVL algorithm hits the termination criteria due to the max developer.

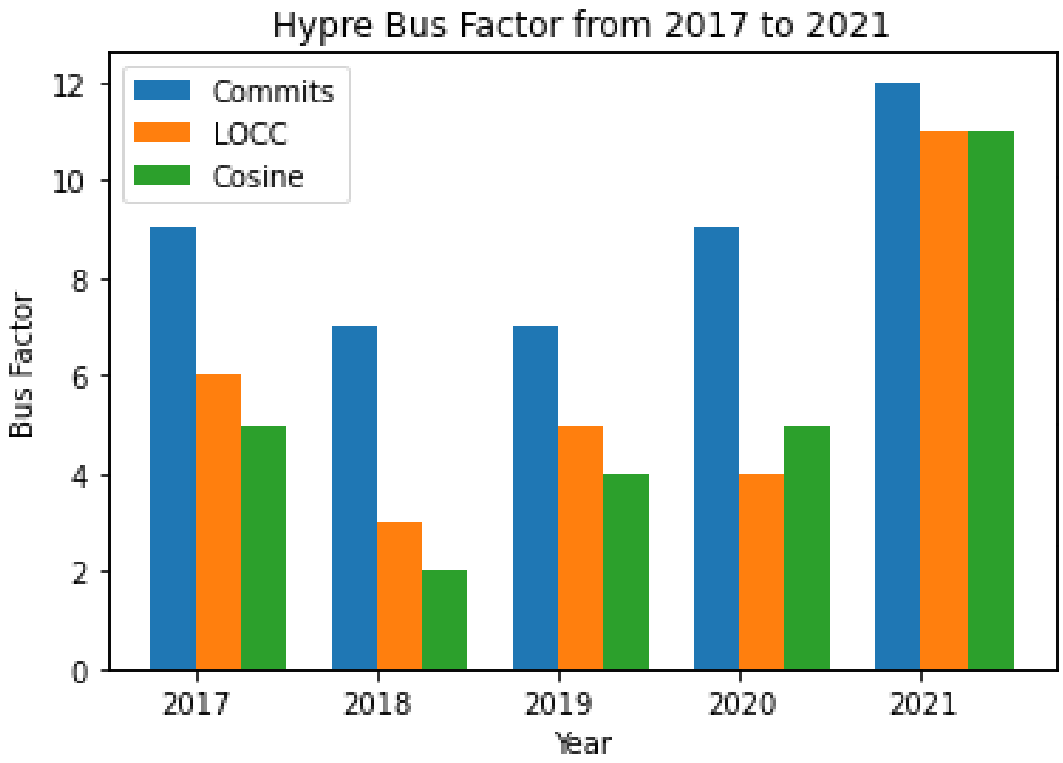

Figure 2- Hypre Bus Factor graph over a five-year period calculated with three different methods, commits, LOCC, and cosine difference.

Through a closer examination of the Hypre BF, the commits always has the highest BF. LOCC and cosine difference have more similar BF values, with the LOCC typically slightly higher. Apart from 2019, the maximum single developer contribution is also greater for commits over LOCC and the cosine difference.

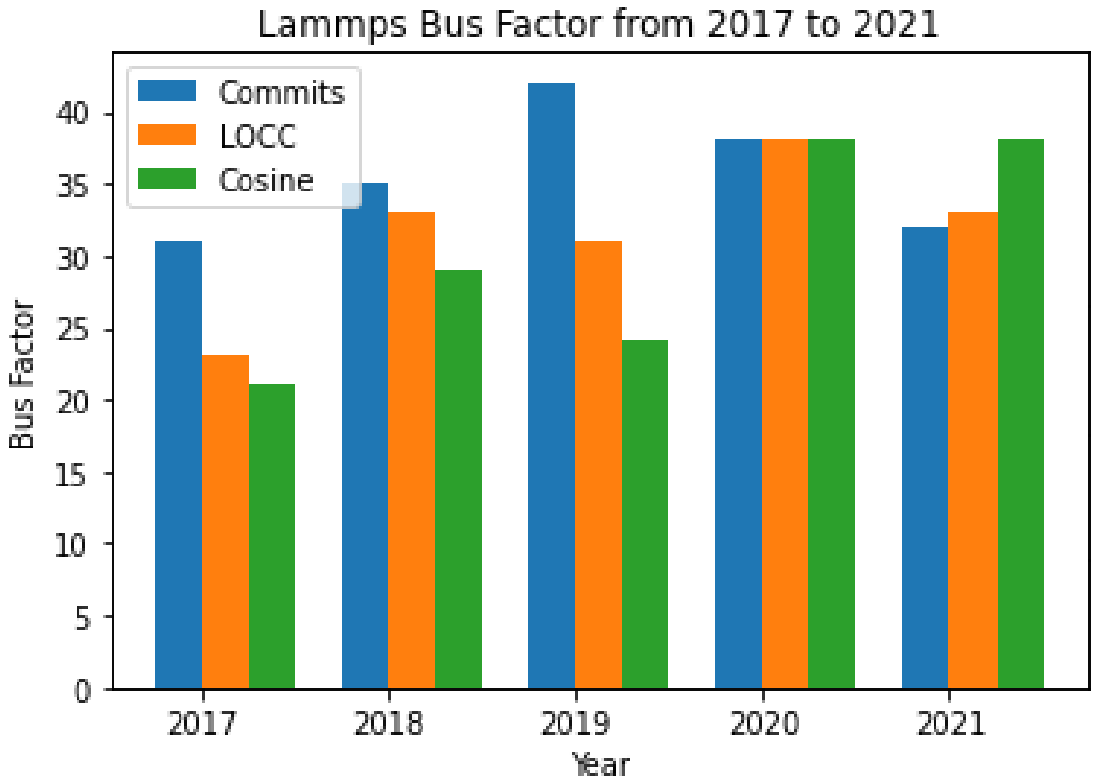

Figure 3 Lammps Bus Factor graph over a five-year period calculated with three different methods, commits, LOCC, and cosine difference.

Lammps follows a similar trend of highest BF for commits, then LOCC, and cosine difference has the lowest BF except for 2020 and 2021. Interestingly, in 2020 the BF was the same across all three metrics, even with the maximum developer value varying.

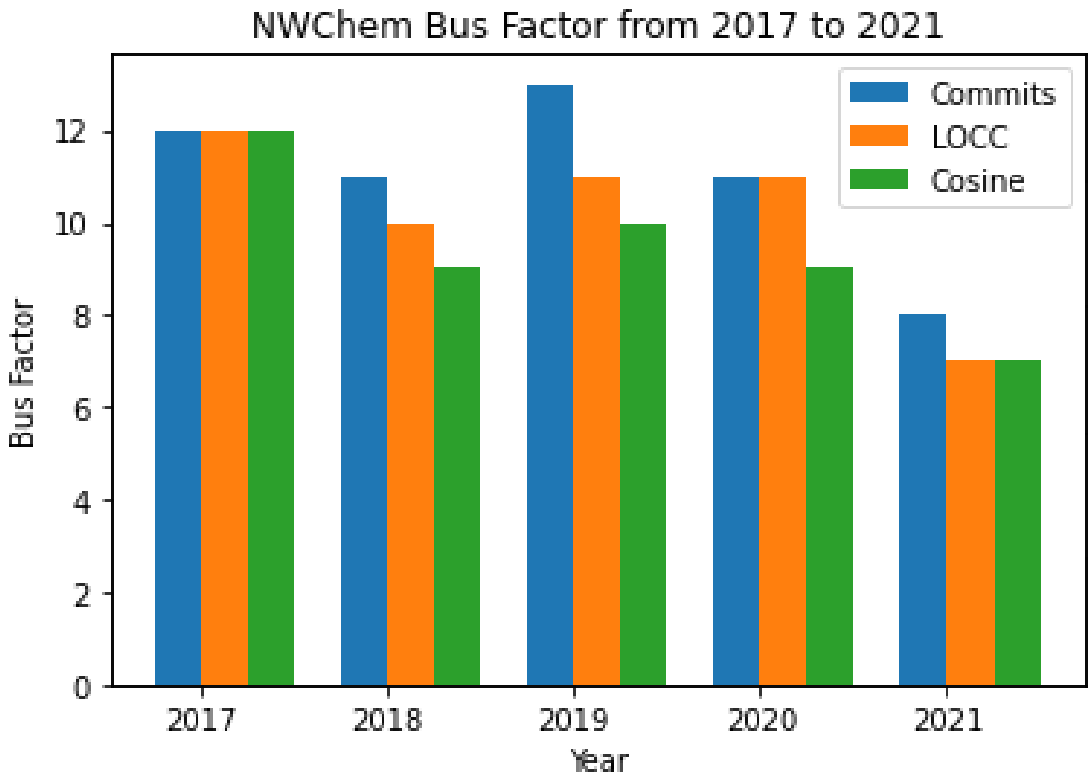

Figure 4 NWChem Bus Factor graph over a five-year period calculated with three different methods, commits, LOCC, and cosine difference.

At first glance, it appears NWChem follows a similar pattern from Table 1 of not having a clear pattern from one metric predicting the BF. However, at a closer glance, there are many more similarities in the BF across each metric. One reason for the more similar BF is due to a much smaller sample of files being run in the algorithm. Additionally, the BF decreases after 2019 even though there is no major change in overall contributions or max developer contributions (Appendix C). Although there is no insight on what is happening internally with the project development, the decreasing BF is slightly concerning due to two reasons. First, most of the project knowledge distributed across too few developers creating potential issues if one of them were to leave the project in the future. And second, the project stakeholders may develop a false sense of stability within the project and make structural changes leading to not enough recourses or people in the project in the future.

CONCLUSIONS

When evaluating the accuracy of the AVL BF algorithm across different metrics, there are a couple important factors to consider. First, the BF of commits is largely dependent on the developers coding style. As discussed earlier, two developers may contribute the same amount of work to a file, but one may do it in two commits, whereas the other might do it in ten. These coding style differences will result in a different degree of authorship per the AVL algorithm and potentially alter the BF. For this reason, when comparing the BF of the lines of code contributed to a file or the cosine difference of the lines of code will always reveal a more accurate BF than commits. I believe the higher degree of accuracy of the LOCC and cosine difference metric correlates to the two metrics having similar BF and value for the maximum contributions across different years and projects.

The second factor that needs to be considered when examining the BF results is that LOCC and the cosine difference are metrics that measure numerical data. The numerical values generated from code files do not reveal whether the code is functional or even used. Therefore, these metrics do not have any indication of quality of work, simply the quantity of work a developer contributed.

The Bus Factor is important for managers when deciding how much manpower is needed on a software project. For the project to have the most success, there needs to be at least the same number of developers as the BF. Projects with higher BF are more stable due to having less chances for more issues due to greater human resources in the project. The commit BF value almost always being higher than the other two metrics will give managers a false sense of security in the stability of a project. For this reason, in order to help reduce potential setbacks

and issues in software projects due to lack of people, it is important to calculate the BF not using commits.

FUTURE RESEARCH

The cosine difference of text is one of many text comparison methods. Cosine difference is not designed to compare code. However, there are several other existing open-source packages available to more in depth string comparisons. Among these is the Levenshtein distance algorithm. This algorithm is used amongst computer science instructors to detect plagiarism. The Levenshtein distance algorithm calculates the number of edits it takes to transform a string into another (Mayank 2019). This algorithm is better than the cosine difference because it can detect higher amounts of similarity between code that goes undetected in LOCC. I believe in order to find the most accurate BF measurement, more metrics used in classroom environments to detect plagiarism should be used when estimating the BF.

Further, the projects examined in this experiment are all open-source research projects. Although some of the projects included contributor guidelines, the fact that anyone can contribute to the project allows many different coding styles which can have a greater influence on the BF. If possible, in a future project, I think open-sourced projects should be compared to similar size closed-source projects from companies that have stricter coding style protocols to see if there is insight on the impact of a regulated coding style on preventing errors.

APPENDIX

Appendix A: Hypre Bus Factor and Max Developer Table

Year	Metric	Bus factor	Max Developer
2017	commits	9	455
	locc	6	321
	cos	5	328
2018	commits	7	60
	locc	3	11
	cos	2	6
2019	commits	7	205
	locc	5	241
	cos	4	253
2020	commits	9	135
	locc	4	24
	cos	5	18
2021	commits	12	771
	locc	11	540
	cos	11	555

Appendix B: Lammps Bus Factor and Max Developer Table

Year	Metric	Bus factor	Max Developer
2017	commits	31	895
	locc	23	226
	cos	21	138
2018	commits	35	1742
	locc	33	677
	cos	29	571
2019	commits	42	1300
	locc	31	283
	cos	24	244
2020	commits	38	1730
	locc	38	1105
	cos	38	958
2021	commits	32	4519
	locc	33	1009
	cos	38	978

Appendix C: NWChem Bus Factor and Max Developer Table

Year	Metric	Bus factor	Max Developer
2017	commits	12	134
	locc	12	98
	cos	12	99
2018	commits	11	197
	locc	10	189
	cos	9	187
2019	commits	13	4664
	locc	11	4551
	cos	10	4555
2020	commits	11	163
	locc	11	160
	cos	9	155
2021	commits	8	156
	locc	7	159
	cos	7	157

REFERENCES

Coplien, James, and Neil Harrison. 2004. *Organizational Patterns of Agile Software Development*. Prentice Hall PTR.

Cosentino, Valerio, Jordi Cabot, and Javier Luis Canovas Izquierdo. March 2015. "Assessing the Bus Facor of Git Repositories." *SANER.* Montreal, Canada.

Ferreira, Mívian Marques, and Marco Tulio Valente. 2017. "A Comparison of Three Algorithms for Computing Truck Factors." *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC).* Buenos Aires, Argentina.

Fritz, Thomas, Gale C. Murphy, Emerson Murphy-Hill, Jingwen Ou, and Emily Hill. March 2014. "Degree-of-Knowledge: Modeling a Developer's Knowledge of Code." *ACM Trans. Softw. Eng. Methodol. 23, 2, Article 14* 42.

Ricca, Filippo, Marco Torchiano, and Alessandro Marchetto. June 2011. "On the Difficulty of Computing the Truck Factor." *Product-Focused Software Process Improvement - 12th International Conference.* Torre Canne, Italy.