Collaboration Strength Metrics and Analyses on GitHub

Natércia A. Batista
Universidade Federal de Minas Gerais
Belo Horizonte, Brazil
natercia@dcc.ufmg.br

Michele A. Brandão
Universidade Federal de Minas Gerais
Belo Horizonte, Brazil
micheleabrandao@dcc.ufmg.br

Gabriela B. Alves
Universidade Federal de Minas Gerais
Belo Horizonte, Brazil
gabrielabrant@dcc.ufmg.br

Ana Paula Couto da Silva
Universidade Federal de Minas Gerais
Belo Horizonte, Brazil
ana.coutosilva@dcc.ufmg.br

Mirella M. Moro
Universidade Federal de Minas Gerais
Belo Horizonte, Brazil
mirella@dcc.ufmg.br

ABSTRACT

We perform social analyses over an important community: the open code collaboration network. Specifically, we study the correlation among features that measure the strength of social coding collaboration on GitHub – a Web-based source code repository that can be modeled as a social coding network. We also make publicly available a curated dataset called GitSED, *GitHub Socially Enhanced Dataset*. Our results have many practical applications such as to improve the recommendation of developers, the evaluation of team formation and existing analysis algorithms.

KEYWORDS

Social Network Analysis, Online Cooperative Work, Tie Strength

ACM Reference format:
Natércia A. Batista, Michele A. Brandão, Gabriela B. Alves, Ana Paula Couto da Silva, and Mirella M. Moro. 2017. Collaboration Strength Metrics and Analyses on GitHub. In *Proceedings of WI '17, Leipzig, Germany, August 23-26, 2017,* 9 pages.
DOI: 10.1145/3106426.3106480

1 INTRODUCTION

Social network analysis has conquered its place as a research area for its importance and increasing community of people. Here, we perform social analyses over an equally relevant community: the open code collaboration network, an important type of social professional network [9]. Specifically, we study the correlation among features that measure the strength of social coding collaboration over a software repository. Indeed, analyzing software repositories allows to answer relevant questions such as "Which patterns do govern developers interactions?", "How do developers sentiment influence their performance?" or "Who are the best developers that a company may contract for a specific job?". Current research has addressed them and others [5, 6, 11, 12, 18, 21–23].

WI '17, Leipzig, Germany

978-1-4503-4951-2/17/08…$15.00
DOI: 10.1145/3106426.3106480

There are different software repositories as SourceForge, GitHub, and Google Code. Here, we mine developers interactions from GitHub[1], a website and online hosting service that offers source code management to users. Currently, GitHub has over 53 million repositories (February 2017) and 14 million users (April 2016)[2]. Many studies have mined GitHub for: modeling reasons for a developer to join a project [22], showing typical practices programmers use to handle exceptions [19], and so on. There are also studies on analyzing the social aspects of software repositories [6, 7, 11, 25]. Studying such aspects may help to understand which ones influence developers productivity [11, 25], and how social interactions between developers and users influence software quality [7].

A specific type of social aspects is the strength of interaction between developers, or the strength of their social tie [2] in the context of social coding [13]. In this context, the strength of interaction has been investigated on GitHub in order to analyze the productivity of developers in projects [11], predict developers collaboration [6] and investigate the acceptance of pull requests [24].

Such studies measure the strength of interactions differently. However, none evaluates the best way to measure such strength, which varies according to number of shared repositories, amount of committed lines, and so on. Also, they do not investigate the correlation among such metrics. Another problem is that, according to Cosentino et al. [12], more than two thirds of the publications (analyzed on their long survey) do not provide the datasets used neither the source code to make the studies replicable. Likewise, as pointed out by Jacobs et al [17], important answers from network analyses come from a good dataset and time-varying relations.

In this paper, we analyze the strength of social collaboration measured by distinct metrics through different programming languages. Our goal is to provide insights about relationships' patterns and their strength over a huge software repository. Our contributions are summarized as follows.

First, we build a dataset called GitSED, for *GitHub Socially Enhanced Dataset*, and make it publicly available, complying to the manifesto by Weller and Kinder-Kurlanda [26]. It provides the following distinguished features: **curated** by being filtered and focusing on two programming languages, **augmented** by adding repository and developer's data not available on GHTorrent [15], and **enriched** with social network information (Section 3).

[1] GitHub: http://github.com
[2] The largest code host on the planet: github.com/features and github.com/about/press

Second, we propose three new metrics for the strength of social coding collaboration considering the commits' number of lines and pontential of contribution. Also, we modify an existing metric for prior social interaction between developers (Section 4).

Third, we evaluate all metrics over two social networks for JavaScript and Ruby (Section 5). Overall, we find that: most developers are active in few repositories, the number of connections between developers varies through different programming languages, and few pairs of developers have interactions in more than one repository. Then, we provide a deep correlation analysis among existing collaboration strength metrics and those proposed here. The results show that the prior social collaboration aspect is related to the commits in shared repositories. Also, commits' number of lines and pontential of contribution are weakly correlated. Such results are relevant to further, deeper and even completely new studies on social coding collaboration strength [6, 11, 24].

2 RELATED WORK

We divide related work in two parts. First, we go over studies that are somehow related to ours. Then, we briefly describe the existing metrics that can be applied to measure collaboration strength.

2.1 Studies Alike

Analyzing GitHub (and software repositories) allows to acquire knowledge that helps to improve software development and, hence, its quality. For example, Dabbish et al. [13] show how to infer developers technical goals and vision from network activities and guess which projects have the chance to be active in long term.

Studying the whole repository reveals general comprehension, whereas studying the communities over specific languages may be even more relevant. Indeed, GitHub studies over specific programming languages reveal patterns of development and collaboration that are peculiar to each language and coding paradigm. For instance, Kery et al. [19] show typical practices programmers use to handle exceptions on Java projects. In a more social perspective, Padhye et al [23] consider 89 popular GitHub projects (and their 108.000+ forks) to study the levels of participation from different communities in the projects and present the results grouped by programming language (11 of the most popular ones).

Still on social perspectives, team formation is also an important topic on developer relationship studies as well. Zihayat at al. [27] propose an efficient algorithm to find teams of experts that cover all the required skills in a project, then maximizing the costs of team formation (like communication cost, expertise cost and the personnel cost). Another example is given by Jarczyk et al. [18] who study task allocation of virtual programmer teams based on real-life data from GitHub repositories and a simulation about the choices of developers of the tasks in which they are going to work.

Other studies consider solutions on distinct areas towards understanding developers and projects relations. For example, Nielek et. al [22] present machine learning algorithms to understand and model the main reasons that lead developers to join specific projects. Likewise, Loyola et. al [21] relate how developers and repositories behave with mutualism (a biological notion in which two species provide benefits to each other).

Finally, a different perspective is to analyze the correlation among distinct features of the software repository. For example, Barnett et al. [5] study the correlation between volume and content of commits messages and tendency to defects in java projects. The results point out metrics for the volume and content of commits message improve Just-In-Time defect prediction models.

Here, we take such studies many steps forward by analyzing the social tie strength over a huge software repository. We start by building a socially enhanced dataset and making it publicly available, not very common in this type of study [26]. We propose new metrics for the strength of social coding collaboration considering features that provide more information on the nature of the tie. Then, we analyze the linear and non-linear correlation among existing metrics and the proposed ones. Overall, our study differs from the existing ones by showing correlated and non-correlated metrics and then revealing a set of metrics that can be used to measure the strength of social coding collaboration.

2.2 Collaboration Strength Metrics

A collaboration SN is mathematically represented by a weighted graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}^w)$, with $\mathcal{V}$ the set of nodes and $\mathcal{E}^w$ the set of non-directed links with an associated weight. Nodes are developers, a link between any two developers exists if both contributed to the same repository, and the link weight measures the strength of the social coding collaboration. Based on this mathematical model, we characterize the nature of the social coding ties using both topological and semantic properties, defined as follows.

2.2.1 Topological Properties. In order to characterize GitHub social network and analyze the correlation of the strength of collaborations with network properties, we use topological metrics based on their ability to represent the relationship between individuals.

Clustering Coefficient (CC). In many networks, if node u connects to node v, and node v to node w, then there is a heightened probability that node u will also be connected to node w. In terms of network topology, transitivity means the presence of a heightened number of triangles in the network, i.e., sets of three nodes connected to each other. Let $T(u)$ be the total number of triangles that u belongs to and $deg(u)$ the degree (number of connected edges) of u, then the clustering coefficient CC of node u is:

$$CC(u) = \frac{2T(u)}{deg(u)(deg(u)-1)},$$

where $T(u)$ is the number of triangles through node u. Such metric shows the tendency of the nodes to cluster together.

Neighborhood Overlap (NO). It measures the neighborhood similarity for any two pair of nodes. Let $\mathcal{N}(u)$ and $\mathcal{N}(v)$ be the set of nodes u and v neighbors, respectively, then

$$NO(u,v) = \frac{|\mathcal{N}(u)| \cap |\mathcal{N}(v)|}{|\mathcal{N}(u)| \cup |\mathcal{N}(v)|}.$$

According to [8] and [14], neighborhood overlap can be used to compute the strength of the ties. Thus, we are interested in analyzing the behavior of such metric in GitHub social network.

Adamic-Adar Coefficient (AA). Given a set of features, it was originally used for computing the similarity between two web pages [1]. In the social network context, the features are the common

neighbors, and the metric is customized as:

$$AA(u,v) = \frac{\sum \forall z \in |\mathcal{N}(u)| \cap |\mathcal{N}(v)|}{log|\mathcal{N}(z)|}.$$

Adamic-Adar coefficient has a similar meaning to neighborhood overlap, because both consider the nodes' neighbors. However, their correlation to other properties is still not known. Also, the denominator indicates unpopular developers (few collaborators) may be more likely to introduce a particular pair of their collaborators to each other. Here, we investigate such aspects and infer its capacity to measure the collaborations strength.

Preferential Attachment (PA). It assumes the likelihood of receiving new edges increases with the node's degree; i.e., the greater the number of neighbors, the higher the value of preferential attachment. To calculate it, we use Networkx [20]: given $\mathcal{N}(u)$ and $\mathcal{N}(v)$ as the sets of neighbors of u and v, respectively, PA is

$$PA(u,v) = |\mathcal{N}(u)||\mathcal{N}(v)|.$$

According to [4], there is a linear relation between the number of neighbors of a node and its P.A. (producing the "rich gets richer" effect). Thus, we investigate such claim in the GitHub network.

Resource Allocation (RA)[3]**.** It measures the resource allocation dynamics on the network. Given a pair of nodes u and v, it represents the amount of resource sent to v by u through their common neighbors. Let $w(u)$, $w(v)$ be the weighted degree of nodes u and v, respectively, and $w(u,z)$, $w(z,v)$, $w(u,v)$ be the weight of links (u,z), (z,v) and (u,v). The weighted degree of a node is the sum of the weight of each edge connected to the node (e.g., a node with two edges with weight 3 has weighted degree as 3+3=6). We compute this metric six times, one for each semantic metric that represents edge weight (defined later), as follows:

$$RA(u,v) = \frac{w(u,v)}{w(u)} + \sum_{z \in \mathcal{N}(u) \cap \mathcal{N}(v)} \frac{w(u,z)w(z,v)}{w(u)w(z)}.$$

Considering the GitHub social network, a developer u is viewed as sending some resource (sharing repositories and committing code) to all of her/his contributors, which has a secondary effect to all of the contributors of the developer v who receives it. No study has used such metric for the strength of social coding collaboration.

Tieness (T). It measures the strength of interactions between individuals [10]. Let $\mathcal{N}(v_i)$ and $\mathcal{N}(v_j)$ be the set of nodes v_i and v_j neighbors, respectively, the metric is calculated as

$$tieness_{i,j} = \frac{\frac{|\mathcal{N}(v_i) \cap \mathcal{N}(v_j)|+1}{1+|\mathcal{N}(v_i) \cup \mathcal{N}(v_j)|-\{v_i,v_j\}} \; ||w_{i,j}||}{2},$$

where $||w_{i,j}||$ is the edge weight of a pair of developers v_i and v_j normalized by unity-based method (it allows to normalize the data within a selected range). Such metric was evaluated for measuring the strength of co-authorship collaboration. Here, we investigate the behavior of tieness in the context of social coding collaboration.

2.2.2 Semantic properties. In some situations, considering only topological properties to measure the strength of ties may not be very accurate, because relationships may be influenced by aspects not only related to the network structure. Then, we consider three semantic properties previously proposed by us to measure collaboration strength (edge weight) [3]. Overall, the semantic properties capture the amount of interaction between two developers. The higher their values, the stronger the interaction between those two. We also propose two new semantic properties to measure the strength of social coding collaboration, described in Section 4.

Number of shared repositories ($SR_{(D_i,D_j)}$). Given two developers $D_i, D_j \in \mathcal{D}$, where $\mathcal{D}$ is the set of all developers in a network, the metric $R_{(D_i,D_j)}$ is the total number of repositories that they both worked at, and is given by the cardinality of $\mathcal{R}$ set (i.e., $|\mathcal{R}|$).

Jointly developers contribution to shared repositories ($JCSR_{(D_i,D_j)}$). Given two developers D_i and D_j and their repositories r_i, the jointly contribution is :

$$JCSR_{(D_i,D_j)} = \frac{\sum\limits_{\forall r_i \in \mathcal{R}} JCSR_{(D_i,D_j,r_i)}}{|\mathcal{R}|}.$$

For example, given r_1 that is only shared by developers A and B, their jointly contribution to r_1 ($JCSR_{(A,B,r_1)}$) is equal to 1. On the other hand, given r_2 that is shared by developers A, B and C, the jointly contribution by A and B to r_2 ($JCSR_{(A,B,r_2)}$) is 0.66. If A and B share only r_1 and r_2, the jointly contribution given by them to these repositories ($JCSR_{(A,B)}$) is 0.83.

Jointly developers commits to shared repositories ($JCOSR_{(D_i,D_j)}$). Given $NC_{(D_i,r_j)}$ as the total number of commits by D_i into repository r_j, $NC_{(D_j,r_j)}$ as the total number of commits by B into repository r_j, and $NC_{(r_j)}$ the total number of commits by any developer into repository r_j. $JCOSR_{(D_i,D_j)}$ is defined as:

$$JCOSR_{(D_i,D_j)} = \sum_{\forall r_i \in \mathcal{R}} \frac{(NC_{(D_i,r_i)} + NC_{(D,r_i)})}{NC_{(r_i)}}.$$

3 METHODOLOGY

Our methodology has the following steps. First, we build a new dataset GitSED based on extracted data from GHTorrent [15], which is an open project that provides GitHub databases. Such dataset is curated, augmented and enriched with social features in Section 3.1. Next, we build a social network that is formed by developers and their projects in Section 3.2. Then, we measure the strength of social coding collaboration with existing topological and semantic properties (as explained in Section 2.2) and new ones proposed in Section 4. Finally, our experimental evaluation analyzes the correlation among collaboration strength metrics in Section 5.

3.1 GitSED: GitHub Socially Enhanced Dataset

Dataset Description. In order to build the dataset, we first extract data from GHTorrent [15]. As pointed out by Cosentino et al in their seminal survey on analyzing GitHub [12], GHTorrent is the most popular way for collecting data from GitHub. Unlike most of the studies covered in their survey (almost 70%) and given the importance emphasized by [26], we make our dataset publicly available at http://www.dcc.ufmg.br/~mirella/projs/apoena.

In summary, the objective of GHTorrent is to monitor public data available on GitHub. We collected a complete dataset on September 15th, 2015. Initially, there was a total of 1,987,760 projects (32 GB of

[3] Also known as Propagation Coefficient (PC)

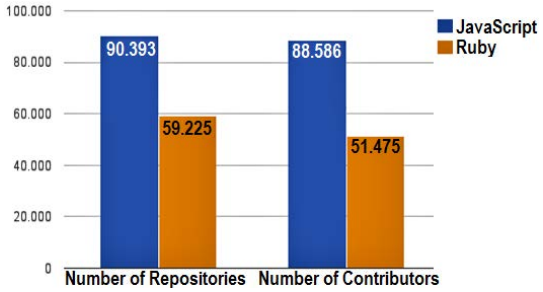

Figure 1: Number of Repositories and Contributors

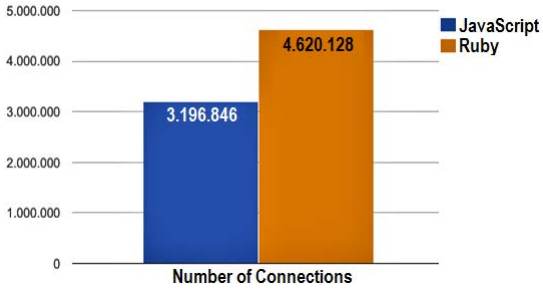

Figure 2: Number of Connections (edges)

data). From those projects, 1,204,212 were forks[4]. We then removed the forked repositories because the changes made on them must be approved by the base repositories (done through a *pull request*)[5]. As such approval is not guaranteed and forked repositories may be used just for testing performance and other side analyses, we work only with non-forked projects, then resulting in 529,405 projects. Even with such cuts, the remaining dataset was still large to process and would need high computation power to handle all data.

Curating the Data. To prune the dataset and make it easier to handle, we focus on two programming languages; without loss of generality and following previous work (Section 2). First, we consider JavaScript with 90,363 repositories (17% of non-forked projects), as it is the most common language on GitHub according to our pre-analysis of the original dataset downloaded from GHTorrent. Second, we include Ruby with 59,225 repositories. Finally, we model a graph as described in Section 2.2, and Figures 1 and 2 show the dataset statistics for both programming languages.

Augmenting with Further Features. After curating (filtering) the original dataset, we have pre-evaluated to identify what kind of analyses such a dataset allows. Moreover, as pointed out by Aiello et al [2], investigating the nature of the social interactions given by the social ties is also important. However, our pre-evaluation showed that such an investigation would be very hard to do over the curated dataset. Hence, we also add other features to allow better analyses of the strength of social coding metrics.

First is the number of lines with which a developer has contributed to a repository, exemplified as follows. Consider two different developers: one who works hours to commit at the end, and another who works the same number of hours but commits at each 10 minutes. Clearly, the importance or relevance of their coding cannot be solely measured by the number of commits.

Nonetheless, GHTorrent does not share the number of lines in each commit. Therefore, we coded a crawler using BeautifulSoup and Requests (modules of Python). The crawler works fine except for those repositories that are not available anymore to be collected (as such numbers were crawled months after collecting the original dataset from GHTorrent). Then, we synchronize GitSED to include developers interactions only from those repositories found by the crawler. Hence, GitSED considers the interactions from 45,926 repositories: 28,584 for Javascript and 17,342 for Ruby.

Second is time. We add an attribute *create_date* defined by the minimum date between *created_at* from GHTorrent and the first commit date. Likewise, *end_date* is the last commit date in the repository. However, a pre-analysis of such creation dates returns commits with this date before the creation of the repository, i.e., a conundrum. Thus, we fix this problem by considering the first commit date as a start date of such repositories. Furthermore, we consider all repositories (even the oldest ones) to allow studying developers relationship over time. Finally, such dates enable to define repositories' duration by adding a field *duration_days*.

Third is statistics of commits: the amount of commits (*number_commits*), sum of added and deleted lines (*number_add_lines* and *number_del_lines*) for each commit, and the number of "committers" in the repository (*number_committers*). The latter represents the developers who effectively contributed for each repository, i.e. not necessarily the original number of members in the repository.

Enriched Relational Schema. We now describe the relational schema to store and handle GitSED in Figure 3. Some tables have processed, enriched data as detailed next.

User. Since GitHub users may collaborate with projects and repositories from different programming languages, we consider all users available and their respective data. Thus, the users in GitSED may code beyond JavaScript and Ruby. Furthermore, the users' information in GitSED 2015 can be expanded by including other data such as gender; then allowing relevant diversity analyses as in [25].

Developer_Social_Network. This table represents all developers interactions for each repository. For each repository in which two developers code, such a pair adds value to it; i.e, for each pair of developers, we quantify such value by considering their number of commits, added and deleted lines in the repositories. We also add the date of start and end of such interaction. Both dates allow to compute the interaction time for each pair of developers, and may be useful for time-oriented complex network analyses.

Social_Network_Metric. We also add social value to the dataset by incorporating social metrics. Specifically, combining attributes allows to measure strength of interaction between developers by different metrics. In its current version (GitSED 2015), this table has fields that represent the social coding analysis metrics proposed by [3]. Additionally, GitSED 2015 allows to compute other social network metrics, specially the topological ones.

[4]Copy of a repository that allows changes without affecting the original project.
[5]Changes committed to an external repository must be approved in the base repository

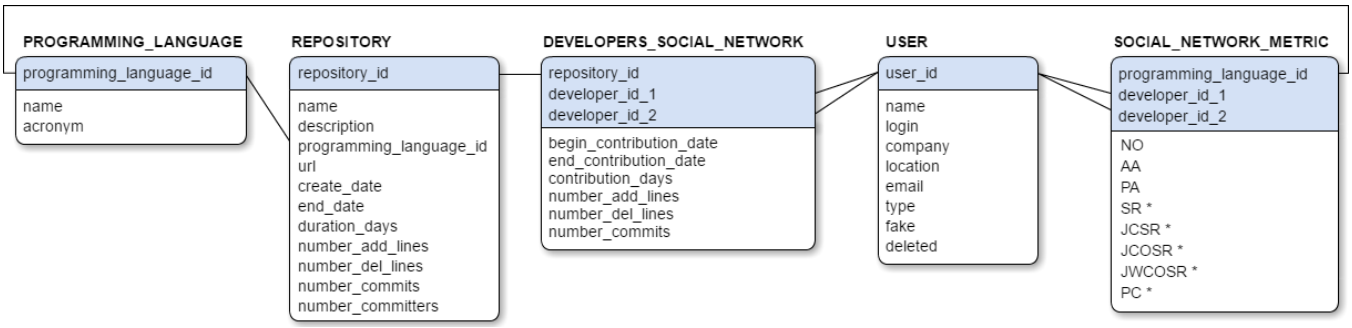

Figure 3: Dataset schema: table names, identifiers (primary keys), attributes and foreign keys (lines connecting tables). *Metric also combined with tieness (T) and resource allocation (RA), i.e., the table has two extra columns for each combination.

3.2 Network Model

The GitHub social coding collaboration network is represented by a weighted graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}^w)$ formalized in Section 2.2. Our curated dataset, with its social characteristics, provides some interesting insights. As expected, the final network is formed by several cliques, as each repository forms one. Such cliques are connected by developers who have coded in more than one repository. Although JavaScript SN has more repositories and nodes, Ruby SN is more dense. This may indicate that Ruby developers tend to contribute to distinct repositories more than JavaScript developers.

4 NEW SEMANTIC PROPERTIES

We now introduce three new metrics for the strength of social coding collaboration. Complementing our previous analyses [3], we focus on understanding how the importance of shared commits, in terms of number of lines modified on each repository, as well as past collaborations and collaboration opportunity may influence the strength of social coding collaboration. Such metrics are based on Granovetter's theory [16], which claims that the strength of relationships is defined as a merge of the time, the emotional force, the intimacy, and the reciprocal services that represent a link between people. Indeed, our new metrics consider the time and reciprocity of the relationships (collaboration) between developers.

4.1 Jointly developers weighted commit to shared repositories

Based on $JCOSR_{(D_i,D_j)}$ features, we propose a new semantic metric, called the jointly developers weighted commit to shared repositories ($JWCOSR_{(D_i,D_j)}$), to differentiate the amount of collaboration between developers. The idea is to weight the commits with their total number of updated lines, then giving more importance to the commits that define more changes to the repository content. Moreover, we consider the following operations for defining updates.

NL$_a$: We consider only the number of lines added in a commit, even if the developer deletes one or more lines. The intuition is only new pieces of code matters for measuring collaboration.

NL$_{a+d}$: We consider the sum of the number of lines added and deleted in a commit, then considering new functionalities introduced on the code as well as possible code corrections.

NL$_{a-d}$: In GitHub, modifications in a code are not distinguished in a commit. For instance, if a developer changes the name of a variable and commits it, GitHub shows 1 line added and 1 line deleted, which indicates a small contribution. The NL$_{a-d}$ metric subtracts the number of deleted lines from the added ones and represents the new contribution of developers to a code by disregarding simple modifications. Also, it considers that the process of adding lines is more expensive than the process of deleting lines. Thus, it is possible to give more importance to the added lines.

As a simple extension, we also consider its module value |NL$_{a-d}$|: a positive value even when the number of deleted lines is greater than the number of added lines. In this case, we consider a developer may have optimized the code (ergo, more deleted lines).

For a pair of developers (D_i, D_j), $JWCOSR_{(D_i,D_j)}$ is given by:

$$JWCOSR_{(D_i,D_j)} = \frac{\sum\limits_{\forall r_i \in \mathcal{R}} \sum\limits_{\forall c_j \in C_{(D_i,r_i)}} CL_{(D_i,c_j)} + \sum\limits_{\forall r_i \in \mathcal{R}} \sum\limits_{\forall c_l \in C_{(D_j,r_i)}} CL_{(D_j,c_l)}}{\sum\limits_{\forall r_i \in \mathcal{R}} CL_{(r_i)}},$$

where $C_{(D_i,r_i)}$ and $C_{(D_j,r_i)}$ are the set of commits made by D_i and D_j in repository r_i, respectively; $CL_{(D_i,c_j)}$ and $CL_{(D_j,c_l)}$ are the contribution, in number of lines, made by the developer D_i and D_j when performing the commit c_j and c_l, respectively (such contributions are calculated using one of the aforementioned operations); and $CL_{(r_i)}$ is the total number of lines modified in r_i, for all commits in r_i, despite a particular developer.

Roughly speaking, $JWCOSR_{(D_i,D_j)}$ calculates the amount of interaction between pairs of developers. Usually, the amount of commits, issues, comments or pull requests are considered to measure the strength of such interaction [11, 24]. Hence, $JWCOSR_{(D_i,D_j)}$ goes further by considering not only the commits, but also the number of lines from each commit.

4.2 Previous Collaboration

Authors in [11] show that developers preferentially join projects in which past social connections are present. Based on their conclusions, we define a new metric that accounts for *previous collaboration*. Hence, $PC_{(D_i,D_j,t)}$ is defined as the amount of collaboration performed by developers D_i and D_j at time t:

$$PC_{(D_i,D_j,t)} = \frac{\sum_{\forall r_i \in \mathcal{R}} \frac{1}{ND_{(r_i,t)}}}{|\mathcal{R}|},$$

where $ND_{(r_i,t)}$ is the total number of developers working in repository r_i at time t, before developer D_j joins it. High values for fraction $1/ND_{(r_i,t)}$ means that D_i is more likely to work with D_j, and vice-versa. For example, if there is only one person in that project, there is only one possible collaboration; whereas, the more people in the project, the more options to choose from and, hence, the less possibility to collaborate with one particular developer. In other words, there is a higher possibility to establish a collaboration if the developer attention is not split over many potential choices.

4.3 Local and Global Potential Contributions

Collaboration opportunities may be improved if a pair of developers work together at many common repositories for large periods of time. Then, given developers D_i and D_j, we coin the metric *local potential contribution*, named $LPC_{(D_i,D_j)}$ as:

$$LPC_{(D_i,D_j)} = \frac{\sum_{\forall r_i \in \mathcal{R}} \frac{T_{(D_i,D_j,r_i)}}{T_{(r_i)}}}{|\mathcal{R}|},$$

where $T_{(D_i,D_j,r_i)}$ is the time interval both developers contribute to repository r_i, defined as the difference between the first commit given by D_i or D_j and the last commit on r_i; and $T_{(r_i)}$ is the interval between the first and last commits on repository r_i.

Local potential contribution may introduce bias, mainly to repositories with small lifetime. For instance, let two repositories r_1 and r_2 with $T_{r_1} = 1$ month and $T_{r_2} = 12$ months, and $A, B, C, D \in \mathcal{D}$. Developers A and B only work together in r_1 for $T_{(A,B,r_1)} = 1$ month. Developers C and D only work together in r_2 for $T_{(C,D,r_2)} = 12$ months. Then, $LPC_{(A,B)} = 1$ and $LPC_{(C,D)} = 0.5$. Applying only local information suggests that developers A and B have more opportunities to collaborate. However, as developers C and D work together more time, it may result in a more prolific contribution. For understanding the collaboration strength over all pairs of developers, it should be more interesting to have a global view of all repositories and their collaborations.

To overcome such drawback, we extend $LPC_{(D_i,D_j)}$ to its global version, named global potential collaboration ($GPC_{(D_i,D_j)}$):

$$GPC_{(D_i,D_j)} = \frac{\sum_{\forall r_i \in \mathcal{R}} T_{(D_i,D_j,r_i)}}{\max_{\forall (D_i,D_j) \in \mathcal{D},\ r_i \in \mathcal{R}} T(D_i,D_j,r_i)}.$$

$GPC_{(D_i,D_j)}$ metric considers the maximum lifetime over all project repositories. Regarding the previous example, if the largest collaboration time interval is 6, $GPC_{(A,B)}$ drops to ≈ 0.16 and $GPC_{(C,D)}$ increases to 1. Then $GPC_{(A,B)}$ metric enables to rank potential contribution strength over all possible contributions inside a network. Although local metrics tend to be easier to calculate, we also apply $GPC_{(D_i,D_j)}$ in our analyses (Section 5.2).

5 ANALYSES AND CORRELATIONS

We now describe the options for weighting the commits (Section 5.1) and the correlation analyses between properties (Section 5.2). Then, we consider the least correlated metrics and present a ranking to the collaborations according to their strength (Section 5.3).

5.1 Weighted Commit Analyses

We now analyze the update operations in the repositories. The goal is to select the best option for weighting the commits to differentiate the amount of collaboration between developers for the cases using $JWCOSR_{(A,B)}$. We consider: the best option gives more information about collaborations than the others; and for different weight metrics, the best has lower computational cost.

Figures 4 (a) and (b) depict Pearson and Spearman correlations for JavaScript and Ruby. Stronger correlations are reached when Spearman coefficient is used, suggesting a non-linear correlation between update operations. Then, as updating measures are correlated and $|\mathrm{NL}_{a-d}|$ consider both deleted and added lines, we apply it on the collaboration strength analysis, next.

5.2 Collaboration Strength Analyses

The strength of social coding collaboration can be measured in different ways with distinct goals [6, 11, 24]. Thus, it is important to identify which aspects better represent such strength. Such analysis is, for example, crucial for building a computational model (or framework) to better classify the strength of ties in order to predict new collaborations, study team formation, analyze information exchange, and so on. Then, our goal here is to identify network properties that better classify the strength of collaboration on GitHub, i.e., we need to define a set of properties that add new information to such a model. We do so by analyzing the properties described in Sections 2 and 4, and selecting those least correlated to each other. In other words, for highly correlated properties, we may choose only one; whereas low correlated properties qualify different data features and should be considered together. Note such study is only possible because of our new dataset GitSED.

To evaluate the correlation, we use both Pearson and Spearman correlation coefficients but, due to space constraints, we only present the results for the former and discuss their differences. Also, we have previously analyzed the correlation among all the properties, and here go over the results grouped by the most (least) correlated ones. Thus, Figures 5a and 5b show the most correlated properties for JavaScript, and Figures 6a and 6b the most correlated properties for Ruby; whereas Figures 5c and 6c show the least correlated properties for JavaScript and Ruby, respectively.

Note that tieness (T) and resource allocation (RA) consider the edges weight represented by the semantic properties. For example, T_SR is tieness with edge weight given by number of shared repositories, and RA_JCSR is resource allocation with edge weight given by JCSR, and so on. Also, there are six possible combinations of tieness and resource allocation with other semantic properties. We emphasize that for all topological properties considered here (Section 2.2), only tieness and resource allocation combine the social network topological characteristics with the semantic ones.

As for the results, considering the correlation between properties in Figures 5 and 6, we do not observe significant difference between the results for JavaScript and Ruby. This may indicate a similar behavior between both programming languages. Furthermore, Figures 5a and 6a show each combination of tieness with

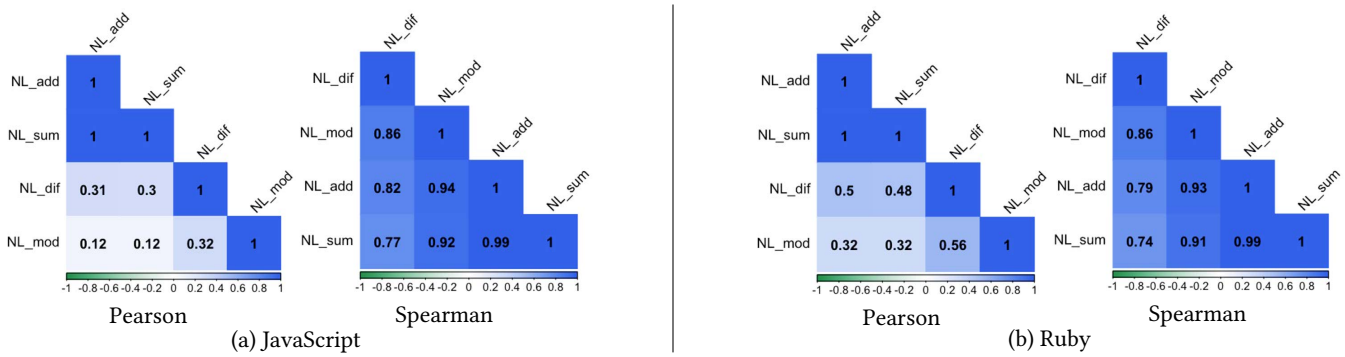

Figure 4: Correlation of the number of lines operations for JavaScript and Ruby

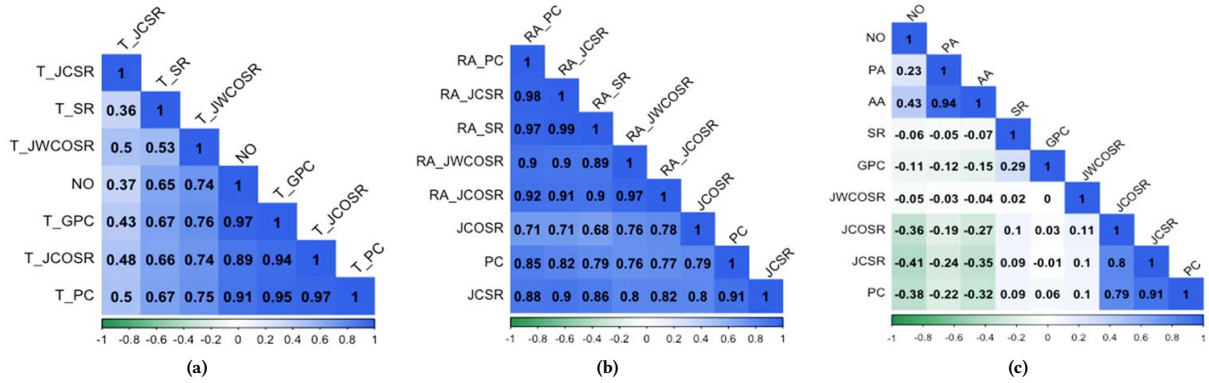

Figure 5: Javascript: Pearson correlation coefficients of the most and least correlated properties.

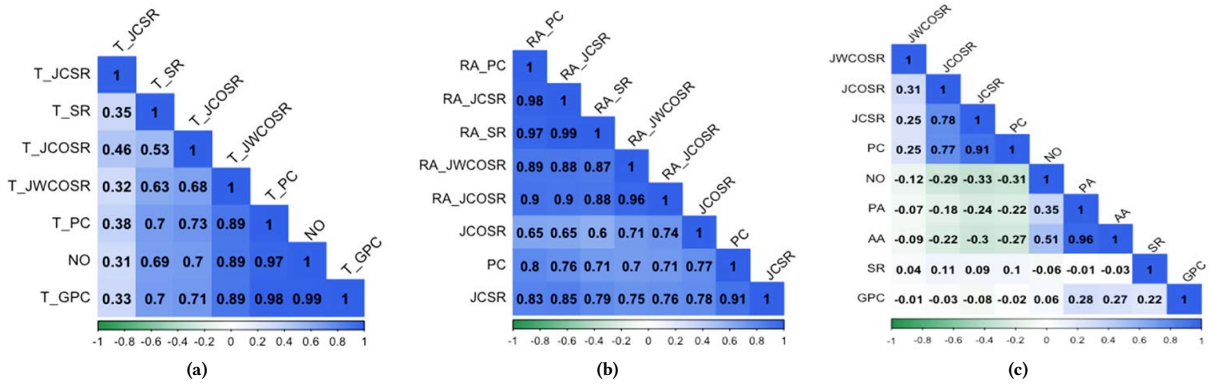

Figure 6: Ruby: Pearson correlation coefficients of the most and least correlated properties.

different semantic properties is largely correlated with each other. The exception is when the edge weight of tieness is *JCSR*, in which *T_JCSR* is moderately correlated with the other metrics. Note that tieness is also largely correlated with neighborhood overlap for different edge weight. This result indicates that is not necessary to consider tieness with all weights to measure the strength of collaboration between developers. As most of them are strongly correlated, we can choose one (*T_SR*, *T_JCOSR*, *T_JWCOSR*, *T_PC* or *T_GPC*) to represent tieness with edge weight in a computational model and *T_JCSR*, which is the least correlated with the others. It is also important to choose the semantic property with low computational cost, which has data easily accessible. For example, the data to compute *SR* is easier gathered than to calculated *JWCOSR*. Thus, it is better to choose *SR* to the weight of tieness than *JWCOSR*.

Table 1: Top-10 pairs of developers ranked by T_JWCOSR followed by the other properties values.

#	JavaScript							Ruby						
	D_1	D_2	T_JWCOSR	T_PC	T_GPC	NO	AA	D_1	D_2	T_JWCOSR	T_PC	T_GPC	NO	AA
1	001	002	0.500	0.500	0.258	0.333	1.443	020	025	0.500	0.333	0.250	0.667	2.485
2	002	006	0.485	0.375	0.250	0.333	1.443	021	027	0.484	0.300	0.063	0.667	2.485
3	003	007	0.281	0.417	0.250	0.333	1.443	021	024	0.442	0.375	0.250	0.667	2.485
4	004	003	0.273	0.375	0.250	0.333	1.443	022	025	0.395	0.312	0.250	0.667	2.485
5	005	008	0.268	0.333	0.258	0.833	4.085	022	028	0.394	0.312	0.250	0.667	2.485
6	005	009	0.267	0.278	0.254	0.833	4.085	020	022	0.391	0.312	0.250	0.667	2.485
7	001	006	0.266	0.375	0.250	0.333	1.443	023	029	0.369	0.375	0.063	0.333	1.443
8	005	010	0.266	0.273	0.254	0.833	4.085	024	027	0.353	0.300	0.063	0.667	2.485
9	005	011	0.266	0.292	0.258	0.833	4.085	025	024	0.346	0.333	0.250	0.667	2.485
10	005	012	0.266	0.275	0.254	0.833	4.085	026	030	0.345	0.500	0.251	0.333	1.443

Figures 5b and 6b show similar patterns of correlations. The resource allocation values are strongly correlated with each other when considering different edge weights. Also, such metric is strongly correlated with other semantic properties, such as *JCOSR*, *PC* and *JCSR*. As RA has higher computational cost, it can be left out from the collaboration strength computation. These results are also similar to Spearman correlation coefficient.

Figures 5c and 6c show the metrics least correlated with each other. Adamic Adar and preferential attachment are inverse correlated in a small and moderate way with *JCOSR*, *JCSR* and *PC*, and these three metrics are largely correlated with each other. Thus, a computational model could consider only one to measure the strength of collaboration. Also, these five metrics are insubstantially correlated with *SR*, *GPC* and *JWCOSR*. Regarding *NO*, by considering tieness, it is not necessary to use *NO*.

Overall, these results indicate that a model to measure the strength of collaboration should: consider *T_JCSR* and only one among *T_SR*, *T_JCOSR*, *T_JWCOSR*, *T_PC* or *T_GPC*; only one between *AA* and *PA* (as they are strongly correlated); only one among *JCOSR*, *JCSR* or *PC*; all three metrics *SR*, *GPC* and *JWCOSR*.

5.3 Collaboration Ranking: An Example

To exemplify how the network properties rank developers according to their strength, Table 1 presents the top-10 pairs of (anonymised) developers ranked by their values of *T_JWCOSR* and the correspondent value to the other metrics. Table 1 reveals there are developers with high collaboration with others. For instance, developer 005 is the one who most collaborate in JavaScript, whereas developers 022 and 025 the most collaborative ones in Ruby. Thus, our metrics identify strong ties with potential collaboration profiles.

6 CONCLUSION

We presented a curated, augmented and enriched dataset built from GitHub. It stores coding collaboration networks from Javascript and Ruby, and allows analyzing social aspects, such as social coding links. Its advantages include: easy to use and replicate and direct computation of different social network metrics.

We also introduced two properties to represent the commits' number of lines and potential of contribution, and modified an existing metric for prior social interaction between developers. Then, we analyzed the correlation between those properties to understand how they relate and choose the ones that capture different information about the collaboration strength. Our results showed that from 21 properties (considering *T* and *RA* combined with edges weights), only five have to be considered in a computational model. Also, we obtained similar results for both programming languages.

As future work, we plan to include repositories from different programming languages, and release new versions of GitSED covering them and other pre-computed social metrics, then allowing studies from different perspectives. We also plan to evaluate the robustness of the proposed metrics and to build a full computational model to properly measure collaboration strength.

Acknowledgments. The authors would like to thank the funding agencies CAPES, CNPq and FAPEMIG, as well as the project FAPEMIG-PRONEX-MASWeb.

REFERENCES

[1] Lada A. Adamic and Eytan Adar. 2003. Friends and neighbors on the Web. *Social Networks* 25, 3 (2003), 211 – 230.
[2] Luca Maria Aiello, Rossano Schifanella, and Bogdan State. 2014. Reading the source code of social ties. In *WebSci*. Bloomington, IN, USA, 139–148.
[3] Gabriela B. Alves et al. 2016. The Strength of Social Coding Collaboration on GitHub. In *SBBD*. Salvador, Brazil, 247–252.
[4] Albert-László Barabási and Réka Albert. 1999. Emergence of scaling in random networks. *Science* 286, 5439 (1999), 509–512.
[5] Jacob G. Barnett et al. 2016. The Relationship Between Commit Message Detail and Defect Proneness in Java Projects on GitHub. In *MSR*. Austin, USA, 496–499.
[6] Roman Bartusiak et al. 2016. Cooperation prediction in GitHub developers network with restricted Boltzmann machine. In *ACIIDS*. Vietnam, 96–107.
[7] Nicolas Bettenburg and Ahmed E Hassan. 2010. Studying the impact of social structures on software quality. In *ICPC*. Braga, Portugal, 124–133.
[8] Michele A. Brandão and Mirella M. Moro. 2015. Analyzing the Strength of Co-authorship Ties with Neighborhood Overlap. In *DEXA*. Valencia, Spain, 527–542.
[9] Michele A. Brandão and Mirella M. Moro. 2017. Social professional networks: A survey and taxonomy. *Computer Communications* 100 (2017), 20–31.
[10] Michele A Brandão and Mirella M Moro. 2017. The strength of co-authorship ties through different topological properties. *J. Braz. Comp. Society* 23, 1 (2017), 5.
[11] Casey Casalnuovo et al. 2015. Developer Onboarding in GitHub: The Role of Prior Social Links and Language Experience. In *ESEC/FSE*. Bergamo, Italy, 817–828.
[12] Valerio Cosentino, Javier Luis, and Jordi Cabot. 2016. Findings from github: Methods, datasets and limitations. In *MSR*. Austin, USA, 137–141.
[13] Laura Dabbish et al. 2012. Social coding in GitHub: transparency and collaboration in an open software repository. In *CSCW*. Seattle, WA, USA, 1277–1286.
[14] David Easley and Jon Kleinberg. 2010. *Networks, crowds, and markets: Reasoning about a highly connected world*. Cambridge University Press.
[15] G. Gousios. 2013. The GHTorrent dataset and tool suite. In *MSR*. San Francisco, USA, 233–236.
[16] M. S. Granovetter. 1973. The strength of weak ties. *The American Journal of Sociology* 78, 6 (1973), 1360–1380.
[17] Abigail Z. Jacobs et al. 2015. Assembling thefacebook: Using Heterogeneity to Understand Online Social Network Assembly. In *WebSci*. Oxford, UK, 18:1–18:10.
[18] Oskar Jarczyk et al. 2014. On the Effectiveness of Emergent Task Allocation of Virtual Programmer Teams. In *IEEE/WIC/ACM WI*. Warsaw, Poland, 369–376.
[19] Mary Beth Kery, Claire Le Goues, and Brad A. Myers. 2016. Examining Programmer Practices for Locally Handling Exceptions. In *MSR*. Austin, USA, 484–487.
[20] David Liben-Nowell and Jon Kleinberg. 2003. The Link Prediction Problem for Social Networks. In *CIKM*. New Orleans, USA, 556–559.
[21] Pablo Loyola and In-Young Ko. 2012. Biological Mutualistic Models Applied to Study Open Source Software Development. In *IEEE/WIC/ACM WI*, Vol. 1. Macau, China, 248–253.

[22] Radoslaw Nielek et al. 2016. Choose a Job You Love: Predicting Choices of GitHub Developers. In *IEEE/WIC/ACM WI*. Omaha, NE, USA, 200–207.
[23] Rohan Padhye, Senthil Mani, and Vibha Singhal Sinha. 2014. A study of external community contribution to open-source projects on GitHub. In *MSR*. Hyderabad, India, 332–335.
[24] Jason Tsay, Laura Dabbish, and James Herbsleb. 2014. Influence of Social and Technical Factors for Evaluating Contribution in GitHub. In *ICSE*. Hyderabad, India, 356–366.
[25] Bogdan Vasilescu, Alexander Serebrenik, and Vladimir Filkov. 2015. A Data Set for Social Diversity Studies of GitHub Teams. In *MSR*. Florence, Italy, 514–517.
[26] Katrin Weller and Katharina E. Kinder-Kurlanda. 2016. A manifesto for data sharing in social media research. In *WebSci*. Hannover, Germany, 166–172.
[27] Morteza Zihayat, Mehdi Kargar, and Aijun An. 2014. Two-phase pareto set discovery for team formation in social networks. In *IEEE/WIC/ACM WI*, Vol. 2. Warsaw, Poland, 304–311.