Evaluating and improving projects' bus-factor: a network analytical framework

Sebastiano A. Piccolo*[1][0000−0002−6986−3344], Pasquale De Meo[2][0000−0001−7421−216X], and Giorgio Terracina[1][0000−0002−3090−7223]

[1] Department of DeMaCS, University of Calabria, Rende (CS), Italy
[2] Department of DICAM, University of Messina, Messina (ME), Italy

Abstract. When enough people leave a project, the project might stall due to lack of knowledgeable personnel. The minimum number of people who are required to disappear in order for a project to stall is referred to as bus-factor. The bus-factor has been found to be real and tangible and many approaches to measure it have been developed. These approaches are problematic: some of them do not scale to large projects, others rely on ad-hoc notions of primary and secondary developers, and others use arbitrary thresholds. None of them proposes a normalized measure of the bus-factor. Therefore, in this paper we propose a framework that, by modelling a project with a bipartite graph linking people to tasks, allows us to *1)* quantify the bus-factor of a project with a normalized measure which does not rely on thresholds; and *2)* increase the bus-factor of a project by reassigning people to tasks. We demonstrate our approach on a real case, discuss the advantages of our framework, and outline possibilities for future research.

Keywords: Bus-factor · Network Robustness · Graph Algorithms · Assignment Problem · Heuristics

1 Introduction

The *bus-factor* of a project – also known as *truck-factor* – is informally defined as the minimum number of people that have to disappear (as if they were hit by a bus) before the project stalls because nobody can complete certain tasks [5,14,19] or because of project fragmentation where integration between different modules, teams, and parts of the project becomes difficult [19]. Key people might become unavailable due to promotions, getting new jobs, going on parental leaves, or any other unforeseeable external event. The risk is tangible and empirical studies have shown that many projects suffer of low bus-factor. For instance, Yamashita et al. [25] analyzed a set of 2496 Github projects and found that, in more than 88% of the repositories, the number of core developers is less than 16. Similarly, Avelino et al. [5] considered a sample of 133 Github projects and estimated that 118 projects had a bus-factor lower than 9. There are even projects which rest on just one key person [5, 22].

* Corresponding author: sebastiano.piccolo@unical.it

Measuring the bus-factor of a project is not a trivial task, as establishing when a project stalls would require to consider the exact role of each person in the project, their contribution, and other project-specific factors as well as some subjective ones. For instance, the first estimation procedure proposed in literature [26], required to solve an NP-Hard problem and did not scale to projects with more than 30 people [5,11]. Another measure [10] requires to define primary and secondary developers and relies on two thresholds as inputs. The current state-of-the-art measure [5], assumes that a project stalls when more than 50% of files in the project are abandoned.

These assumptions and thresholds are arbitrary, targeted to computer science projects, and might even be project-specific. Furthermore, quantifying the bus-factor by simply counting the number of people makes the measure dependent on project size, preventing comparisons across different projects. Finally, there is still a lack of methods to improve the bus-factor of a project by reassigning people to tasks. Therefore, in this paper we develop a general graph theoretical framework that allows us to *1)* quantify the bus-factor of a project with a normalized measure that does not rely on arbitrary thresholds and enables meaningful comparisons across projects; and *2)* improve the bus-factor of a project by reassigning people to tasks.

We model a project with a bipartite graph $\mathcal{G}$ where vertices are people and tasks, and edges specify who works on which task. On $\mathcal{G}$, people's unavailability can be simulated by removing the corresponding nodes. As such, the problem of quantifying the bus-factor finds its natural position within Network Science, through the theory of *network robustness* [2, 19, 23]. Consequently, we consider the connected component of $\mathcal{G}$ with the largest number of tasks, and measure the relative fraction of tasks in it as people are removed from $\mathcal{G}$. We quantify the bus-factor as the area under this decay curve normalizing it by its theoretical maximum. To this end, we devise an algorithm which computes the bus-factor in linear time, with respect to the number of people, by using a union-find data structure.

Finally, we take on a crucial issue in project management: increasing the bus factor with the purpose of augmenting the project's chances of success. In this regards, we propose two algorithms, based on hill-climbing and simulated annealing respectively, that optimize our measure of bus-factor by reassigning people to tasks. We demonstrate our framework on a real case and find that we are able to improve the project bus-factor by 40%, in a way that is statistically higher than what we would expect by randomly assigning people to tasks.

In sum, we make the following contributions: *1)* we develop a theoretical framework grounded in Network Science to quantify and increase projects' bus-factor; *2)* we define $\mathcal{B}(\mathcal{G})$, a normalized measure of the bus-factor of a project; *3)* we provide an $\mathcal{O}(n)$ algorithm to compute $\mathcal{B}(\mathcal{G})$; and *4)* we provide two efficient heuristics to optimize $\mathcal{B}(\mathcal{G})$ by reassigning people to tasks.

2 Our framework

In this section, we discuss our framework to compute and increase a project bus-factor. We treat the assignment of people to tasks as a bipartite network and use the topology of such a network in order to compute a bus-factor index. The bus-factor we compute is normalized; therefore, our measure enables comparisons between the bus-factor of projects of different sizes – both in terms of number of people and number of tasks. We begin our exposition with some preliminaries on graph theory. In the following we will use the words graph and network interchangeably.

2.1 Preliminaries on graphs

Networks are a convenient mathematical formalism to represent and analyze complex systems, focusing on the way the components of such systems are connected; components are represented as vertices and their connections are represented with edges between the vertices. Graph methods are flexible enough to enable the representation and analysis of heterogeneous systems [1], with multiple layers of connections [8, 16], which evolve over time [12, 13].

An undirected graph is a pair $\mathcal{G} = (V, E)$ where V is the set of vertices and $E \subseteq V \times V$ is the set of edges. A graph is represented through its adjacency matrix A; with $A_{ij} = 1$ if vertices i and j are connected and $A_{ij} = 0$ otherwise. The degree k_i of a vertex i is the number of vertices connected to it; that is, $k_i = \sum_j A_{ij}$. A *subgraph* of a graph G is the graph induced by a subset of vertices of G. A connected component of a graph is a connected subgraph which is not part of any other connected subgraph. The largest connected component is called giant connected component and we refer to it as GCC. If the set of vertices V can be divided in two sets V_1 and V_2 with $V_1 \cap V_2 = \emptyset$, such that $\forall (i, j) \in E \; i \in V_1 \wedge j \in V_2$ the graph is called *bipartite*. In the remainder of this paper, we denote a bipartite graph as a tuple $\mathcal{G} = (V_1, V_2, E)$.

We use bipartite graphs in order to represent the assignment of people to tasks; in the following section, we develop a measure to evaluate the bus-factor of a project from such an assignment.

2.2 Evaluating the bus-factor of a project from the assignment of people to tasks

Let us recall that the informal definition of bus-factor is the minimum number of people who must disappear before the project stalls. Defining when a project stalls in a formal way is far from easy, because many factors can determine when the project stalls; including the role of each person, the extent of their contributions to the project, and other subjective factors [11, 22, 24]. In particular, prior literature has relied on ad-hoc notions of primary and secondary developers, arbitrary thresholds to establish when a project stalls, and has produced non-normalized measures. To overcome these limitations, we propose a measure

for the bus-factor which, with a single number, summarizes the robustness of a project to people removal and enables comparisons between different projects.

We represent the assignment of people to tasks with a bipartite graph $\mathcal{G} = (P, T, E)$, where P is the set of vertices representing the people involved in the project, T is the set of vertices representing the tasks, and E is the set of edges. Let us denote with TGCC the *task-wise giant connected component*; that is, the connected component which connects the highest number of tasks (nodes in T). To evaluate the robustness of $\mathcal{G}$, we draw upon the concept of network robustness [2]. Starting from $\mathcal{G}$, we can remove in a given order – typically in decreasing order of degree – vertices from P and, each time we remove a person, we measure the fraction of nodes in T that are still connected to the TGCC. The choice of removing people by their degree is motivated by three facts: *1)* real-world networks are resistant against random vertex removal [2, 19]; *2)* vertex degree is generally correlated with other centrality measures and topological properties [17]; and *3)* finding the ordering of nodes which maximizes network disruption is, in general, a problem that can be mapped into the set cover problem, which is NP-Hard. The use of people's degree in decreasing order represents a greedy approximation to the set cover problem.

The people removal procedure describes a decay curve: on the y-axis we have the fraction of tasks still connected to the TGCC and on the x-axis we have the fraction of people removed from the project. We compute the bus-factor as the area under this decay curve, divided by the theoretical maximum.

Definition 1 (Bus-factor). *Given a network $\mathcal{G} = (P, T, E)$, with P the set of n people and T the set of m tasks, the bus-factor of such an assignment is obtained by computing the area under the decay curve of the TGCC as people are removed from the network in decreasing order of their degree. We denote the non-normalized bus-factor of a network $\mathcal{G}$ as $B(\mathcal{G})$ and compute it as follows:*

$$B(\mathcal{G}) = \frac{1}{2n} \sum_{q=1}^{n} [S(\mathcal{G}, q-1) + S(\mathcal{G}, q)] \tag{1}$$

Where $S(\mathcal{G}, q)$ is the relative number of tasks still connected to the TGCC, after the removal of q people, with $1 \leq q \leq n$. The normalized bus-factor is computed by dividing the non-normalized bus-factor for the theoretical maximum ($\mathcal{B}_n$), which is obtained if $\mathcal{G}$ is a fully connected network (that is, if every person is assigned to every task):

$$\mathcal{B}(\mathcal{G}) = \frac{B(\mathcal{G})}{\mathcal{B}_n} = \frac{1}{2n-1} \sum_{q=1}^{n} [S(\mathcal{G}, q-1) + S(\mathcal{G}, q)] \tag{2}$$

$\mathcal{B}(\mathcal{G})$ is equal to 1 for the fully connected bipartite graph, and equal to 0 for the fully disconnected graph.

Algorithm 1 Fast bus-factor computation

```
1: function Bus-Factor(𝒢, removal_order)
2:     𝒰 ← UnionFind(𝒢.nodes())
3:     S ← [0]
4:     for all p ∈ removal_order.reverse() do
5:         neigs = Γ(𝒢, p) ∪ {p}
6:         𝒰.union(neigs)
7:         S.append(𝒰.tasks_in_TGCC())
8:     end for
9:     S ← S/m
10:    return S.reverse()
11: end function
```

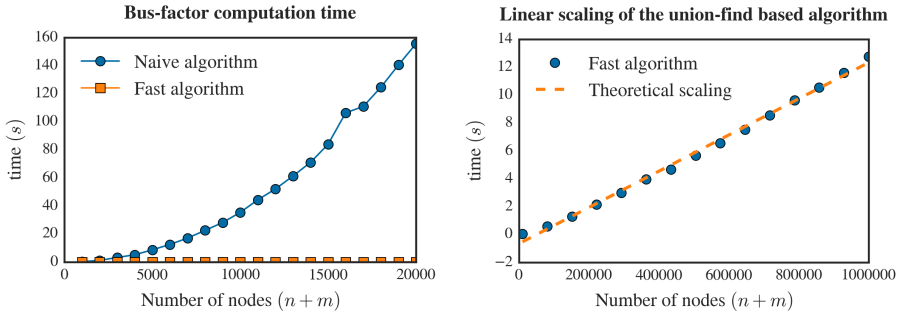
2.3 A fast algorithm to compute the bus-factor

Computing the bus-factor naively results in an algorithm which is quadratic in the number of people. In fact, in order to find the TGCC of a network we need to perform a depth-first search (DFS) to find all the connected components and then we need to find the component with the highest number of tasks. The DFS has a computational complexity equal to $\mathcal{O}(|V| + |E|)$. In order to compute the bus-factor of a project, we need to remove n people and perform the DFS n times. Since in our bipartite graph we have n people and m tasks, $|V| = n + m$. Assuming a sparse network, we have $|E| = \mathcal{O}(|V|) = \mathcal{O}(n + m)$. As such, in a sparse network, the naïve computation of the bus-factor has a computational complexity of $\mathcal{O}(n^2 + nm)$.

Here, we present an algorithm (Algorithm 1) which computes the bus-factor, exactly, in linear time with respect to the number of people. We use a union-find structure and start with n people and m tasks disconnected. We proceed in increasing order of degree from the last person to remove to the first one and we progressively add edges. We keep track of the set with the largest number of tasks in the union-find structure and store the number of tasks in it. Such a set corresponds to the TGCC of the network.[3] In this way, we only perform n union operations and after each union operation we keep track of the number of tasks in the TGCC. Our approach is described in Algorithm 1.

The function to compute the bus-factor takes in input the bipartite graph of the assignment of people to tasks $\mathcal{G}$ and a vector of people in decreasing order of degree (`removal_order`). $\mathcal{U}$ is a union-find structure initialized with all the nodes of $\mathcal{G}$ and S is a vector where we store the number of tasks in the TGCC. For each person p in the vector (`removal_order`), in reverse order, we perform a union operation between the person p and the set of tasks to which p is connected (denoted with $\Gamma(\mathcal{G}, p)$). S is finally divided by the total number of tasks and is returned in reverse order. The output of Algorithm 1 is the relative number of tasks in the TGCC after each removal. In order to compute the normalized bus-

[3] Our approach can be adapted, by keeping track of the largest set in the union-find structure, to compute the robustness of an arbitrary network.



(a) (b)

Fig. 1: **(a)** Time comparison between the naïve algorithm and our fast one, based on a union-find structure. **(b)** Empirical evaluation of the linear scaling of Algorithm 1. The execution time has been measured on 15 Erdos-Renyi bipartite graphs, with connection probability $p = 2\log(2n)/n$. The theoretical scaling line is obtained by fitting a line to the 15 points through ordinary least squares.

factor from S, we need to compute $\mathcal{B}(G)$ by computing the area under S in $[0, 1]$ normalized by $\mathcal{B}_n$, using (2).

In Fig. 1(**a**) we contrast the execution time of our fast algorithm for the computation of the bus-factor against the naïve approach. It is possible to see that the naïve approach becomes impractical even for relatively small networks, and the speedup offered by our algorithm is significant. For our experiment we generated 20 Erdos-Renyi bipartite graphs, with $n = m$ and a total number of nodes $(n + m)$ varying in [1000, 20000] with increments of 1000. We set the connection probability $p = 2\log(2n)/n$; this value ensures that the resulting graphs are both sparse and connected. To better show the performance of our algorithm, we perform a second experiment in which we generate 15 Erdos-Renyi bipartite graphs, as before, with a total number of nodes $(n+m)$ equally spaced in [10000, 1000000], and measure the execution time to compute the bus-factor. We report our results in Fig. 1(**b**). It is possible to confirm that the union-find based algorithm for the bus-factor computation scales linearly with the number of people.

Now that we have both a measure to evaluate the bus-factor of a project and a fast algorithm to do so, we can turn our attention to methods for improving the bus-factor of a project by increasing $\mathcal{B}(\mathcal{G})$. In the next section, we are going to discuss some methods to improve the bus-factor of a project by reallocating people to tasks.

2.4 Increasing the bus-factor of a project: initial models

It is possible to increase the bus factor of a project in many ways. One option is to assign people more responsibilities and, thus, more tasks. This option is

Algorithm 2 Bus-factor optimization with hill-climbing

```
1: function Bus-Factor-Opt-HC(𝒢, n_iter)
2:     x ← 𝒢, H_x ← H(x)
3:     improved ← true
4:     while improved do
5:         improved ← false
6:         for i ∈ 0 ... n_iter do
7:             x' ← 𝒫(x), H_x' ← H(x')
8:             if H_x' < H_x then
9:                 x ← x', H_x ← H_x'
10:                improved ← true
11:            end if
12:        end for
13:    end while
14:    return x
15: end function
```

equivalent to increase the density of the bipartite network, which does increase the bus-factor of the project. However, the risk of such an option is to overload people with tasks and responsibilities making the project more prone to errors and rework [19, 21].

Another option is hiring new people, assigning them strategically to tasks in order to increase the bus-factor of the project. This option is certainly interesting, particularly from the point of view of mathematical optimization, because it involves many variables, a hiring budget, managing the hiring process, and assigning people to tasks in order to optimize the bus-factor. However, this option is impractical and has important drawbacks: the hiring process is conditioned by a budget and can be slow, inefficient, and ineffective [6]. Additionally, adding people to a project might also be detrimental since it imposes higher coordination efforts [9, 18, 20].

Here, we focus on a third strategy. We focus on increasing the bus-factor of a project by reassigning tasks to people, improving the topology of the bipartite network against people removal. We do this by preserving the total number of tasks assigned to each person. In the following, for simplicity and without loss of generality, we make the simplifying assumption that a person can be assigned to any task. In those situations where this is not the case, the previous assumption can be relaxed by incorporating – in the objective function or in the edge swap procedure described in the following – the information about who can perform which task. Reassigning tasks to people while preserving the total number of tasks assigned to each person can be achieved by swapping pairs of edges as follows. Let (p_1, t_1) and (p_2, t_2), with $p_1, p_2 \in P, p_1 \neq p_2$ and $t_1, t_2 \in T, t_1 \neq t_2$ be two edges of an assignment graph $\mathcal{G} = (P, T, E)$; the edge-swap procedure takes (p_1, t_1) and (p_2, t_2) in input and returns the new edges (p_2, t_1) and (p_1, t_2). The swap is valid if it does not produce a double edge; that is, if none of new edges (p_2, t_1) and (p_1, t_2) is already present in $\mathcal{G}$. With such a procedure it is

Algorithm 3 Bus-factor optimization with simulated annealing

```
1: function BUS-FACTOR-OPT-SA(G, t_m, t_M, n_iter, α)
2:     t ← t_M
3:     x ← G, H_x ← H(x)
4:     while t > t_m do
5:         for i ∈ 0 ... n_iter do
6:             x' ← P(x)
7:             H_x' ← H(x')
8:             ΔH ← H_x' − H_x
9:             if ΔH < 0 or rand(0,1) < e^(−ΔH/t) then
10:                x ← x'
11:                H_x ← H_x'
12:            end if
13:        end for
14:        t ← α × t
15:    end while
16:    return x
17: end function
```

possible to rearrange the edges of $\mathcal{G}$ in order to increase the bus-factor measure $\mathcal{B}(\mathcal{G})$. We denote the swapping procedure as $\mathcal{P}(\mathcal{G})$.

The problem of assigning people to tasks in order to maximize $\mathcal{B}(\mathcal{G})$ reduces to the generalized assignment problem, which is NP-Hard [27]. Therefore, here we employ two heuristics that increase the bus-factor of a given project by minimizing $H(\mathcal{G}) = -\mathcal{B}(\mathcal{G})$. The first algorithm is based on an hill-climbing procedure, which is a greedy optimization approach. The hill-climbing algorithm works as follows: first, a solution proposal is generated as $\mathcal{G}' = \mathcal{P}(\mathcal{G})$. If $H(\mathcal{G}') < H(\mathcal{G})$, then the proposal is accepted and $\mathcal{G} = \mathcal{G}'$; otherwise, the proposal is rejected and $\mathcal{G}$ is unchanged. The process is iterated until convergence, when the algorithm reaches an optimum. Since hill-climbing is a greedy optimization strategy, there is no guarantee that the algorithm reaches the global optimum. The final, rewired, $\mathcal{G}$ is the solution; i.e. the optimized assignment network with an increased bus-factor. Our hill-climbing approach is described in Algorithm 2.

The second algorithm is based on simulated annealing, which is a global optimization algorithm [15]. In simulated annealing, a solution proposal is generated as $\mathcal{G}' = \mathcal{P}(\mathcal{G})$. Let $\Delta H = H(\mathcal{G}') - H(\mathcal{G})$ be the bus-factor variation between the new candidate solution and the current one, if $\Delta H < 0$, the solution is accepted – as in the hill-climbing algorithm. If $\Delta H \geq 0$, the proposal is accepted with probability $\mathrm{P}(H) = e^{\frac{-\Delta H}{t}}$, where t is the temperature parameter. The initial temperature t_M is an input of the algorithm. After each iteration, a new temperature is computed as $t = t \times \alpha,\ \alpha \in (0, 1)$. As such, simulated annealing can accept proposals that are worse than the current solution. This simple difference with the hill-climbing algorithm enables simulated annealing not to remain stuck into a local optimum [15]. It is worth noting that an iteration in simulated annealing can involve the generation of more than one solution proposal; thus,

the algorithm can perform more than one edge swap at each iteration [15]. The number of solution proposals to generate at each iteration (`n_iter`) is an input of the algorithm. The algorithm terminates when the temperature t is lower than or equal to a minimum temperature t_m, provided as input. The pseudo-code of this algorithm is reported in Algorithm 3.

Tuning the parameters of the simulated annealing is not an easy task, since the right combination of parameters is problem dependent [7, 15]. Nevertheless, there are some general indications and rules of thumb on how to set the parameters. In general, the higher the number of iterations, the better the final solution is [7]. However, there is a trade-off between the goodness of the solution and the computational time needed to reach it. The temperature should be cooled slowly, since, it has been shown that if temperature is cooled slowly enough, simulated annealing converges to the optimal solution [7]. The cooling factor α is typically chosen in $[0.99, 0.9999]$. The initial temperature, t_M should be chosen in such a way that around 80% of the proposals generated are accepted. The final temperature t_m should be very close to zero; typical values range in $[10^{-6}, 10^{-10}]$. The values of α, t_M, and t_m are modulated by the value of `n_iter`, which sets the number of proposals generated at each temperature t.

3 Experiments and analysis

To demonstrate our framework, we use – as a case study – the bipartite network describing the allocation of people to tasks in the context of a real project of a biomass power plant, from Piccolo et al. [19]. The network consists of 111 people, 148 tasks, and 926 edges between people and tasks. The network exhibits the *small-world* property with density equal to 0.056, clustering coefficient equal to 0.3, and average shortest path length equal to 6.378. On average, a person is connected to 8.34 tasks. The network consists of 3 connected components (Fig. 2A) with a giant connected component containing 107 people and 145 tasks. Piccolo et al. [19], used this bipartite network to show the importance of people connected to a high number of tasks in determining both the assignment robustness, and the response to error propagation phenomena. In the following, we demonstrate our framework in two ways. First, we quantify the bus-factor of the project under examination and we assess its statistical significance with respect to an appropriate null model. Second, we improve the bus-factor of the project by reallocating people to tasks with the two algorithms previously discussed. We assess the statistical significance of the new assignments and evaluate the computational gains provided by our fast (union-find based) algorithm to compute the bus-factor.

3.1 Quantifying the bus-factor and assessing its statistical significance

Algorithm 1 provides us a fast way to compute the bus-factor decay curve. In Fig. 2B, we show the bus-factor decay curve of the project under examination

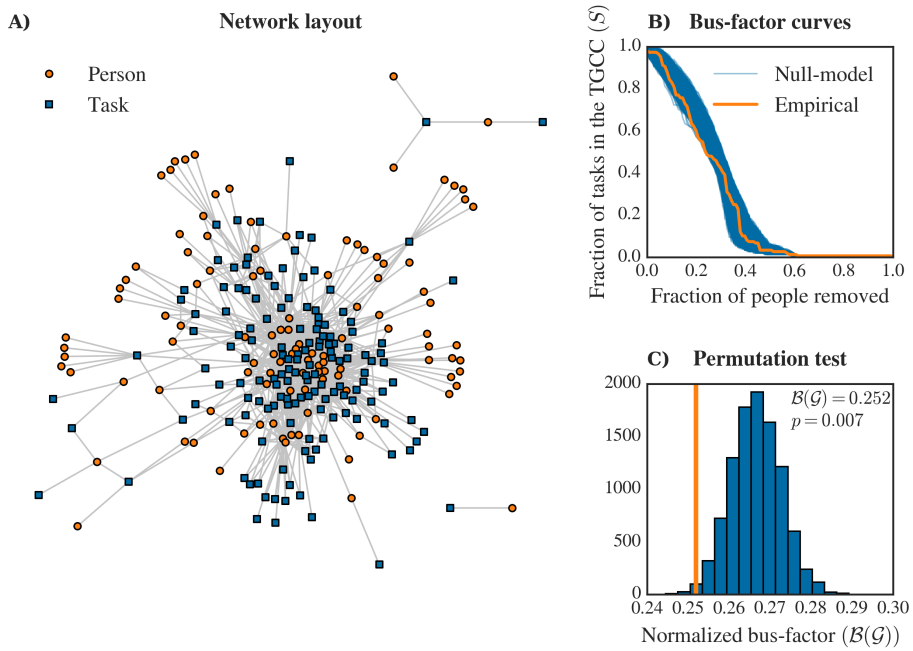


Fig. 2: **A**: Layout of the empirical network. The network consists of three connected components end exhibits modular organization. **B**: Decay curves of the TGCC as people are removed in decreasing order of their degree. The orange line shows the decay of the TGCC of the empirical network, while blue lines are computed on 10000 networks generated with a degree-preserving bipartite null model. **C**: Permutation test which assesses the statistical significance of the bus-factor of the empirical network against the degree-preserving null model. The bus-factor of the empirical network is statistically lower than what we would expect if people were randomly allocated to tasks (p-value $= 0.007$).

in orange. We compare this curve with 10000 random bipartite graphs obtained with a *degree-preserving* null model. The degree-preserving null model fixes the degree of each person and task and connects a random person with a random task, until all the connections have been assigned. In Fig. 2B, we show the bus-factor decay curve of the null model realizations with blue lines. It is evident that all the curves have similar shapes and that the curve of the empirical network is closer to the lower end of the range. Fig. 2B shows the dependence of the bus-factor on the degree distributions of people and tasks. This behaviour is in line with other findings on network robustness [2].

Using formula (2) to compute the normalized bus-factor $\mathcal{B}(\mathcal{G})$, not only allows us to condense the information of the curve in a single number, but it also enables the assessment of the statistical significance of the bus-factor of a project. In fact, the 10000 random bipartite graphs realizations can be seen as random

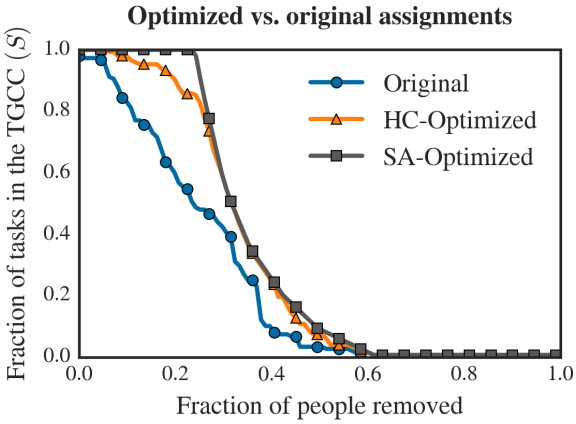


Fig. 3: Original vs. optimized assignments of people to tasks. For the hill climbing algorithm we set the parameter n_iter = 100. For the simulated annealing we set $t_M = 0.5$, $t_m = 10^{-8}$, n_iter = 100 and $\alpha = 0.99$. The bus-factor of the original network is 0.252; the bus-factor of the network optimized with the hill-climbing algorithm is 0.333; the bus-factor of the network optimized with simulated annealing is 0.352.

permutations of the edges of the original network, and can be used in a sort of permutation test. The bus-factor of the empirical network is $\mathcal{B}(\mathcal{G}) = 0.252$. A bus-factor which is 25% of the bus-factor of a fully connected bipartite assignment between 111 people and 148 tasks, with a connection density of 0.056, could appear as remarkable. However, when we compute the normalized bus-factor for all the 10000 random networks, we can see that the bus-factor of the project is actually lower than the bus-factor that we would expect at random. In Fig. 2C, we show the results of the permutation test. The vertical orange line shows the bus-factor of the project under examination and the blue histogram shows the distribution of the bus-factor under the null model. The p-value for the test $\mathcal{B}(\mathcal{G}) < \mathcal{B}(\text{null})$ is 0.007, indicating that the bus-factor of our project is statistically lower than what we would expect by randomly allocating people to tasks. This finding echoes those from prior research about the low bus-factor of many projects [5, 10, 14, 19, 25, 26].

3.2 Improving the bus-factor of the case study project

In this section, we focus on optimizing the bus-factor of the project under examination by rewiring the original assignment network so as to improve its bus factor. To this end, we employ the algorithms defined in Section 2.4. In Fig. 3, we plot the bus-factor decay curves for the original network $\mathcal{G}$ (in blue), the network $\mathcal{G}_{HC}$ obtained by rewiring $\mathcal{G}$ via the hill-climbing algorithm (in orange), and the network $\mathcal{G}_{SA}$ obtained by rewiring $\mathcal{G}$ via the simulated annealing algorithm (in

Table 1: Speedup provided by the fast bus-factor computation over the naïve one, in the optimization algorithms.

Algorithm	**parameters**	$\mathcal{T}_n$ (sec)	$\mathcal{T}_f$ (sec)	**speedup**
Hill Climbing	n_iter = 100	42.3	2.4	17.6×
Hill Climbing	n_iter = 1000	260.8	8.4	31×
Hill Climbing	n_iter = 10000	1600.1	43.9	36×
Simulated Annealing	$t_M = 0.5$ $t_m = 10^{-8}$ n_iter = 1 $\alpha = 0.99975$	1523.5	53.5	28×
Simulated Annealing	$t_M = 1$ $t_m = 10^{-10}$ n_iter = 1 $\alpha = 0.99975$	2017.2	68.9	29×
Simulated Annealing	$t_M = 0.5$ $t_m = 10^{-8}$ n_iter = 100 $\alpha = 0.99$	4015.2	119.6	34×

$\mathcal{T}_n$: computation time with naïve bus-factor computation.
$\mathcal{T}_f$: computation time with fast bus-factor computation (Algorithm 1).

gray). The improvement is evident. Numerically, $\mathcal{B}(\mathcal{G}_{HC}) = 0.333$ which is an improvement of 32% over $\mathcal{B}(\mathcal{G})$; and $\mathcal{B}(\mathcal{G}_{SA}) = 0.352$ which is an improvement of 40% over $\mathcal{B}(\mathcal{G})$. Compared with the statistical ensemble of networks from the degree-preserving null-model, both $\mathcal{B}(\mathcal{G}_{HC})$ and $\mathcal{B}(\mathcal{G}_{SA})$ are statistically higher than the bus-factor we would expect if people were randomly assigned to tasks, with p-value $p < 0.0001$. We note that optimizing the bus-factor as defined in (2), also leads to increasing the number of people that it is necessary to remove in order to disconnect the first task from the assignment network. This can clearly be seen by comparing, in Fig. 3, the point where the decay curves begin to decline. For the network rewired by simulated annealing, it is necessary to remove more than 20% of people (by their degree) in order to disconnect the first task from the TGCC. In contrast, after removing the same number of people from the original network, more than 50% of tasks result disconnected from the TGCC. The network found by simulated annealing is probably optimal. We experimented with different sets of parameters (see Table 1) and each instance of simulated annealing converged to the same final network. In contrast, the hill-climbing algorithm did not converge to the same network found by simulated annealing. However, when we set n_iter = 10000, hill-climbing finds a network only slightly sub-optimal, with a bus-factor ~ 0.352, very close to the one found by simulated annealing. In Table 1, we report the execution time for hill-climbing and simulated annealing under different parameters choices. We

also contrast the execution time when we use the naïve quadratic algorithm to compute $H(\mathcal{G})$ with the execution time when we use Algorithm 1 to compute $H(\mathcal{G})$. The use of Algorithm 1 to compute the objective function, renders both hill-climbing and simulated annealing about 30 times faster than their counterparts which use the naïve algorithm. We also observe that simulated annealing with the linear algorithm to compute the objective function is faster or as fast as the hill-climbing algorithm which uses the quadratic algorithm to compute the objective function. Table 1 also shows the importance of tuning the parameters of simulated annealing in order to avoid wasted computation. Taken together, these results demonstrate the goodness of our framework both from a theoretical standpoint, in that they quantify the bus factor of a project and enable its optimization, and from an algorithmic standpoint, in that they provide efficient algorithms for measuring and improving the bus factor. Finally, our framework allows us to assess the statistical significance of the bus-factor of a project by comparing it with what we would expect if people were randomly allocated to tasks.

4 Related works

The bus-factor is mainly covered in computer science literature by considering two related problems: *1)* finding the *core* developers – i.e. developers who contribute to the vast majority of the code base – and *2)* quantifying the bus-factor of a project. Core developers are detected by estimating the *degree of authorship* (DoA) of each file by combining factors extracted from repositories, code review, meetings, etc [5, 10, 14, 25]. These approaches are specific to computer science and, since in this paper we propose a general framework, we do not review them. We just point out that our framework can be paired with these approaches; for instance, by using the DoA to build the assignment graph $\mathcal{G}$.

Regarding the estimation of the bus-factor, Zazworka et al. [26] proposed the first approach, based of finding the minimal set X of people that would belong to more than t% (where t is an arbitrary threshold) of files; the bus-factor would be estimated as the size of X. Their approach was criticized for making too strong assumptions [5, 11] and being non-scalable [11, 22]. A second approach comes from Cosentino et al. [10]: a set of metrics to estimate the bus-factor based on the notions of primary and secondary developers, detected according to their contribution to the code base. When a certain number of files is abandoned, that is no primary or secondary developer is linked to it, the project is stalled. Avelino et al. [5] used the DoA to link developers to files and proposed the following heuristic to estimate the bus-factor: they iteratively remove the person associated with the highest number of files and increase the bus-factor by one, until more than 50% of the files are abandoned. A comparative analysis from Ferreira et al. [11] showed that the heuristic from Avelino et al. [5] is the best performing one. This heuristic is still the base for more recent approaches, such as the one from Jabrayildaze et al. [14]. A different angle is taken by CSDETECTOR [3, 4]: a C4.5 algorithm, which detects so called *community smells*, including the bus-

factor. However, csDetector makes binary predictions and does not estimate the bus-factor; it requires it as input.

Compared with existing heuristics, our measure $\mathcal{B}(\mathcal{G})$ offers several advantages: *1)* it does not use arbitrary thresholds; *2)* it is normalized; thus, it enables comparisons across different projects; *3)* it can be computed in linear time, scaling to very large networks. Finally, since our measure summarizes the decay of the TGCC, it quantifies nuances that prior approaches cannot take into account. For instance, as Fig. 3 shows, $\mathcal{G}_{HC}$ and $\mathcal{G}_{SA}$ remain with a TGCC lower than 50% after the same number of people have been removed. According to the heuristic from Avelino et al., these networks have the same bus-factor. However, this is not the case: the decay curve of $\mathcal{G}_{SA}$ Pareto-dominates the decay curve of $\mathcal{G}_{HC}$.

5 Conclusions and future work

The *bus-factor* evaluates the robustness of projects against people unavailability. Prior literature has shown that many projects have low bus-factor and that consequences of a low bus-factor can be quite severe. As such, a number of methods have been proposed to *1)* formally quantify the bus-factor of a given project; and *2)* detect the core people in a project. These approaches, however, rely on some assumptions and thresholds to define when a project stalls, which have been criticized as arbitrary. Furthermore, these approaches use non-normalized measures, making comparisons across projects difficult.

Therefore, in this paper, we proposed a framework grounded in network science to *1)* quantify the bus-factor with a normalized measure; and *2)* assign people to tasks in order to maximize the bus-factor of a project. We developed a linear algorithm to evaluate the robustness of a project and measure the bus factor. We showcased our approach using the network of allocation of people to tasks from a real design project. We have shown that our framework allows us to compare the bus-factor of different projects and we used this property to assess the statistical significance of the case project against a null model where people are randomly allocated to tasks. Finally, we developed two algorithms to reassign people to tasks in order to improve the bus-factor, obtaining an improvement of 40% over the case project.

We plan to extend this research in the following ways: *1)* with respect to the theory of bus-factor, we plan to characterize the hardness of measuring the bus-factor as well as assigning people to tasks in order to maximize it. *2)* with respect to the algorithms, we aim to expand our assignment algorithms by taking into account other factors such as the suitability of people to perform certain tasks. Furthermore, we will investigate other ways to remove people from a project in order to obtain better estimates of the bus-factor. *3)* with respect to practice, we plan to test our framework on a set of Github repositories comparing our measure with existing ones.

Acknowledgments. This research is partially supported by MUR under PNRR project PE0000013-FAIR, Spoke 9 - Greenaware AI – WP9.2.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Agreste, S., De Meo, P., Ferrara, E., Piccolo, S., Provetti, A.: Analysis of a heterogeneous social network of humans and cultural objects. IEEE Transactions on Systems, Man, and Cybernetics: Systems **45**(4), 559–570 (2015). https://doi.org/10.1109/TSMC.2014.2378215
2. Albert, R., Jeong, H., Barabási, A.L.: Error and attack tolerance of complex networks. nature **406**(6794), 378–382 (2000)
3. Almarimi, N., Ouni, A., Chouchen, M., Mkaouer, M.W.: Improving the detection of community smells through socio-technical and sentiment analysis. Journal of Software: Evolution and Process **35**(6), e2505 (2023). https://doi.org/https://doi.org/10.1002/smr.2505
4. Almarimi, N., Ouni, A., Mkaouer, M.W.: Learning to detect community smells in open source software projects. Knowledge-Based Systems **204**, 106201 (2020). https://doi.org/https://doi.org/10.1016/j.knosys.2020.106201
5. Avelino, G., Passos, L., Hora, A., Valente, M.T.: A novel approach for estimating truck factors. In: 2016 IEEE 24th International Conference on Program Comprehension (ICPC). pp. 1–10 (2016). https://doi.org/10.1109/ICPC.2016.7503718
6. Behroozi, M., Shirolkar, S., Barik, T., Parnin, C.: Debugging hiring: what went right and what went wrong in the technical interview process. In: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Society. p. 71–80. ICSE-SEIS '20, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3377815.3381372
7. Bertsimas, D., Tsitsiklis, J.: Simulated annealing. Statistical science **8**(1), 10–15 (1993)
8. Boccaletti, S., Bianconi, G., Criado, R., Del Genio, C.I., Gómez-Gardenes, J., Romance, M., Sendina-Nadal, I., Wang, Z., Zanin, M.: The structure and dynamics of multilayer networks. Physics reports **544**(1), 1–122 (2014)
9. Brooks Jr, F.P.: The mythical man-month (anniversary ed.). Addison-Wesley Longman Publishing Co., Inc. (1995)
10. Cosentino, V., Izquierdo, J.L.C., Cabot, J.: Assessing the bus factor of git repositories. In: 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER). pp. 499–503 (2015). https://doi.org/10.1109/SANER.2015.7081864
11. Ferreira, M., Valente, M.T., Ferreira, K.: A comparison of three algorithms for computing truck factors. In: 2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC). pp. 207–217 (2017). https://doi.org/10.1109/ICPC.2017.35
12. Holme, P., Saramäki, J.: Temporal networks. Physics reports **519**(3), 97–125 (2012)
13. Holme, P., Saramäki, J.: Temporal network theory, vol. 2. Springer (2019)
14. Jabrayilzade, E., Evtikhiev, M., Tüzün, E., Kovalenko, V.: Bus factor in practice. In: Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice. p. 97–106. ICSE-SEIP '22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3510457.3513082

15. Kirkpatrick, S., Gelatt Jr, C.D., Vecchi, M.P.: Optimization by simulated annealing. science **220**(4598), 671–680 (1983)
16. Kivelä, M., Arenas, A., Barthelemy, M., Gleeson, J.P., Moreno, Y., Porter, M.A.: Multilayer networks. Journal of complex networks **2**(3), 203–271 (2014)
17. Li, C., Li, Q., Van Mieghem, P., Stanley, H.E., Wang, H.: Correlation between centrality metrics and their application to the opinion model. The European Physical Journal B **88**, 1–13 (2015)
18. Parraguez, P., Piccolo, S.A., Perišić, M.M., Štorga, M., Maier, A.M.: Process modularity over time: modeling process execution as an evolving activity network. IEEE transactions on engineering management **68**(6), 1867–1879 (2019)
19. Piccolo, S.A., Lehmann, S., Maier, A.: Design process robustness: a bipartite network analysis reveals the central importance of people. Design Science **4**, e1 (2018)
20. Piccolo, S.A., Lehmann, S., Maier, A.M.: Different networks for different purposes: A network science perspective on collaboration and communication in an engineering design project. Computers in Industry **142**, 103745 (2022)
21. Piccolo, S.A., Maier, A.M., Lehmann, S., McMahon, C.A.: Iterations as the result of social and technical factors: empirical evidence from a large-scale design project. Research in Engineering Design **30**(2), 251–270 (2019)
22. Ricca, F., Marchetto, A., Torchiano, M.: On the difficulty of computing the truck factor. In: Proceedings of the 12th International Conference on Product-Focused Software Process Improvement. p. 337–351. PROFES'11, Springer-Verlag, Berlin, Heidelberg (2011)
23. Schneider, C.M., Moreira, A.A., Andrade Jr, J.S., Havlin, S., Herrmann, H.J.: Mitigation of malicious attacks on networks. Proceedings of the National Academy of Sciences **108**(10), 3838–3841 (2011)
24. Torchiano, M., Ricca, F., Marchetto, A.: Is my project's truck factor low? theoretical and empirical considerations about the truck factor threshold. In: Proceedings of the 2nd International Workshop on Emerging Trends in Software Metrics. p. 12–18. WETSoM '11, Association for Computing Machinery, New York, NY, USA (2011). https://doi.org/10.1145/1985374.1985379
25. Yamashita, K., McIntosh, S., Kamei, Y., Hassan, A.E., Ubayashi, N.: Revisiting the applicability of the pareto principle to core development teams in open source software projects. In: Proceedings of the 14th International Workshop on Principles of Software Evolution. p. 46–55. IWPSE 2015, Association for Computing Machinery, New York, NY, USA (2015). https://doi.org/10.1145/2804360.2804366
26. Zazworka, N., Stapel, K., Knauss, E., Shull, F., Basili, V.R., Schneider, K.: Are developers complying with the process: an xp study. In: Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement. ESEM '10, Association for Computing Machinery, New York, NY, USA (2010). https://doi.org/10.1145/1852786.1852805
27. Özbakir, L., Baykasoğlu, A., Tapkan, P.: Bees algorithm for generalized assignment problem. Applied Mathematics and Computation **215**(11), 3782–3795 (2010). https://doi.org/https://doi.org/10.1016/j.amc.2009.11.018