A Comparative Study of Algorithms for Estimating Truck Factor

Mívian Ferreira*, Guilherme Avelino*, Marco Tulio Valente*, Kecia A. M. Ferreira†
*Federal University of Minas Gerais, Belo Horizonte, Brazil
Email: {mivian.ferreira,gaa,mtov}@dcc.ufmg.br
†Federal Center for Technological Education of Minas Gerais, Belo Horizonte, Brazil
Email: kecia@decom.cefetmg.br

***Abstract*—In modern software projects, it is crucial to have reliable data about how knowledge on the source code is distributed among the team members. This information can help for example to avoid "islands of knowledge" and to prevent the risks associated to the loss of key developers. Truck factor is a key measure proposed to estimate such risks. Basically, truck factor (aka bus factor) designates the minimal number of developers that have to be hit by a truck (or quit) before a project is incapacitated. Although being a key measure of the concentration of information among team members, we still have few algorithms proposed to estimate truck factors. More importantly, we lack rigorous comparisons of the existing algorithms. Therefore, in this paper we provide a comparative study of the two main algorithms proposed in the literature to estimate truck factors. For this purpose, we rely on a large dataset of 133 popular GitHub systems. We compare both the performance of these algorithms and the truck factors estimated by them.**

***Index Terms*—truck factor; code authorship; github**

I. Introdução

A rotatividade de membros de uma equipe de desenvolvimento colaborativo de software pode ser um grande risco para a sobrevivência a longo prazo de um projeto, uma vez que podem existir nessa equipe desenvolvedores que centralizam o conhecimento sobre o código-fonte do software [1]. Esse fato torna relevante obter dados sobre como o conhecimento está distribuído entre os desenvolvedores que participam de um projeto de software. Entretanto, obter essa informação ainda é um desafio em Engenharia de Software.

O conceito de autoria de código vem sendo utilizado em estudos que tem por objetivo fornecer dados que sejam capazes de elucidar essa questão [2]. Uma aplicação emergente do conceito de autoria de código-fonte é a estimativa de *truck factors*. De acordo com Bowler [3], *truck factor* é, em uma tradução livre, "o número de pessoas em seu time que devem ser atropeladas por um caminhão antes que o projeto entre em sérios problemas". Segundo Ricca et al. [4], essa medida pode ser utilizada para identificar possíveis riscos no projeto, assinalando o quanto o projeto é dependente de determinados desenvolvedores; o custo de substituir um determinado desenvolvedor no projeto; e de que forma o conhecimento está distribuído entre os membros da equipe de desenvolvimento.

Embora *truck factor* seja um conceito importante para o gerenciamento de projetos de desenvolvimento de software, existem poucos trabalhos relacionados ao tema. Em seu trabalho, Zazworka et al. [2] apresentam um algoritmo força-bruta para o cálculo de *truck factor*. Entretanto, essa abordagem é, em seu pior caso, um problema NP-difícil [5]. Além disso, tal algoritmo considera que qualquer desenvolvedor que realizou um *commit* em um determinado arquivo fonte faz parte de sua lista de autores. No presente artigo, o algoritmo de Zazworka será denominado algoritmo ZWK e seu método para cálculo de autoria será denominado de autoria por *commit*.

Mais recentemente, Avelino et al. [6] propõem um novo algoritmo para estimativa de *truck factors*, que se baseia em uma heurística gulosa. Além disso, em sua proposta, os autores adaptam para cálculo de autoria de arquivos fonte uma métrica originalmente proposta para mensurar conhecimento sobre código fonte [7], [8]. No presente artigo, o algoritmo de Avelino será denominado de algoritmo AVL e seu método para cálculo de autoria será denominado de autoria por DOA (*degree of authorship*).

Neste trabalho, apresenta-se um estudo comparativo entre os algoritmos ZWK e AVL. Para isso, usa-se um *dataset* extenso com 133 sistemas GitHub populares, incluindo sistemas implementados em seis linguagens de programação distintas. Além disso, o algoritmo ZWK é avaliado usando tanto seu método original para definição de autoria (autoria por *commit*) como o método de autoria por DOA. Por outro lado, o algoritmo AVL é avaliado considerando seu método original para estimativa de autoria (autoria por DOA) e o método de autoria por *commit*. Para realizar a comparação proposta, quatro questões de pesquisa são investigadas e discutidas:

- (Q1) Qual o impacto do método de cálculo de autoria (autoria por *commit* vs. autoria por DOA) no tempo de execução dos algoritmos ZWK e AVL?
- (Q2) Qual o impacto do método de cálculo de autoria nos valores de truck factor produzidos pelos algoritmos ZWK e AVL?
- (Q3) Os tempos de execução dos algoritmos ZWK e AVL são significativamente diferentes?
- (Q4) Existem diferenças nos resultados de truck factor produzidos pelos algoritmos ZWK e AVL?

O restante deste trabalho está organizado da seguinte forma: Seção II apresenta os algoritmos avaliados neste trabalho; Seção III relata a metodologia utilizada para avaliação dos algoritmos; Seção IV apresenta os resultados obtidos; Seção V

apresenta uma breve discussão sobre os achados do trabalho; Seção VI discute as ameaças à validade do trabalho; Seção VII apresenta trabalhos relacionados e Seção VIII traz as conclusões e indicações de trabalhos futuros.

II. Algoritmos para Estimativa de Truck Factors

Nesta seção são apresentados os dois algoritmos para estimativa de *truck factors* comparados e avaliados neste artigo: Algoritmo ZWK, proposto por Zazworka et al. [2] e Algoritmo AVL, proposto por Avelino et al. [6].

A. Algoritmo ZWK

A primeira descrição de um algoritmo para o cálculo de *truck factor* foi apresentada por Zazworka et al. [2]. Nesse trabalho os autores fazem uso da métrica para apontar diferenças entre a distribuição de conhecimento em projetos que fazem uso de *Extreme Programming* (XP) e projetos que não utilizam essa metodologia. Embora apresentem a lógica empregada, os autores não disponibilizam a real estrutura desse algoritmo. A primeira descrição de uma implementação para esse algoritmo, é apresentada por Ricca et al.[9] (Algoritmo 1).

Algoritmo 1: Algoritmo de Zazworka et al.(2010)

```
   Input: int cobertura, lista devs, lista arquivos
   Output: int truck factor do sistema
1  begin
2      for j = 1 to devs.size() do
3          coberturaMinima ← 100;
4          for each comb ∈ Combinacao(j, devs) do
5              cobertos ← 0;
6              for each arquivo ∈ arquivos do
7                  if conjDevs(arquivo) − comb ≠ null
                    then
8                      cobertos ← cobertos + 1;
9                  end
10             end
11             coberturaAtual = cobertos/arquivos.size() *
                100;
12             if coberturaAtual ≤ coberturaMinima then
13                 coberturaMinima ← coberturaAtual;
14             end
15         end
16         vetorCoberturaMinima[j] ← coberturaMinima;
17         if vetorCoberturaMinima[j-1] < cobertura then
18             break;
19         end
20     end
21     return j;
22 end
```

No Algoritmo ZWK original, um desenvolvedor é considerado autor se realizou qualquer modificação (commit) em um arquivo do sistema. Baseado nessa premissa, o algoritmo gera a combinação dos n autores tomados j a j, sendo $1 \leq j \leq n$. Para cada combinação obtida, calcula-se o número de arquivos cobertos (*i.e.,* arquivos que permaneceriam com autores considerando a saída dos desenvolvedores presentes no conjunto, linhas 6-10). A cobertura do sistema (*coberturaAtual*, linha 11) é o percentual de arquivos que ainda permanecem com pelo menos um autor no conjunto de desenvolvedores do sistema. O *truck factor* é dado pelo número de desenvolvedores presentes na primeira combinação cuja cobertura (*coberturaMinima*) seja menor que o valor determinado como cobertura.

B. Algoritmo AVL

Em um trabalho recente, Avelino et al. [6] propõem uma heurística gulosa para o cálculo de *truck factor*. Nessa abordagem, a autoria é calculada por meio da fórmula de *degree-of-authorship* (DOA), originalmente proposta por Fritz et al. [7], [8]. Para cada arquivo de código-fonte presente no sistema a ser avaliado, é calculado o grau de autoria dos desenvolvedores que já realizaram pelo menos um *commit* no arquivo em questão. Um dado desenvolvedor é considerado autor do arquivo se possuir, após a normalização dos valores, $DOA > 0,75$. Diferente da abordagem de análise combinatória apresentada por Ricca et al. [9], a heurística desenvolvida por Avelino et al. [6] é baseada na retirada do desenvolvedor com maior autoria de arquivos no sistema.

Algoritmo 2: Heurística Avelino et al. (2016)

```
   Input: Mapeamento "A" de autores e arquivos
   Output: int truck factor do sistema
1  begin
2      S ← getArquivos(A);
3      tf ← 0;
4      while autores ≠ ∅ do
5          cobertura ← getCobertura(S, A);
6          if cobertura < 0.5 then
7              break;
8          end
9          autores ← removeTopAutor(A);
10         tf ← tf + 1;
11     end
12     return tf;
13 end
```

Para calcular o *truck factor* de um sistema, a heurística recebe como parâmetro uma lista contendo o mapeamento $autores \rightarrow arquivo$. Baseado nos arquivos e autores obtidos através dessa mapeamento, é realizado o cálculo da cobertura atual do sistema. Esse cálculo identifica a proporção de arquivos que ainda possuem autores. Por exemplo, valores de cobertura acima de 0.5 indicam que mais de 50% dos arquivos tem pelo menos um autor. Sendo assim, realiza-se a retirada do desenvolvedor que possui autoria no maior número de arquivos do sistema. As retiradas são realizadas de forma iterativa até que mais de 50% dos arquivos fiquem descobertos ou não existam mais autores a serem removidos. O pseudocódigo dessa abordagem é apresentado no Algoritmo 2.

III. Projeto do Estudo Comparativo

A. Dataset

Neste estudo comparativo, utilizou-se o mesmo dataset proposto por Avelino et al. [6] para validar os resultados obtidos pelo seu algoritmo com desenvolvedores GitHub. O dataset inclui sistemas desenvolvidos nas seis linguagens com maior número de repositórios no GitHub: JavaScript, Python, Ruby, C/C++, Java e PHP. Inicialmente, para criação do dataset, foram selecionados os 100 sistemas mais populares de cada uma das seis linguagens. O critério de popularidade adotado foi o número de estrelas que o sistema possui no GitHub.[1]

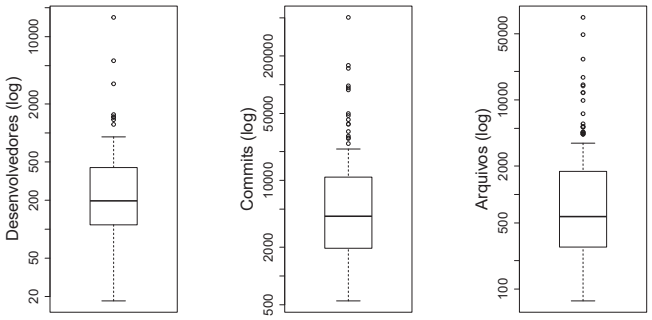
Dessa seleção inicial, foram filtrados os sistemas mais importantes em cada linguagem, com time de desenvolvedores, histórico de desenvolvimento e tamanho (número de arquivos) significativos. Para tal, partiu-se do conjunto completo de repositórios inicialmente selecionados e, considerando os sistemas de uma dada linguagem por vez, foram descartados aqueles que se encontravam no primeiro quartil da distribuição em pelo menos uma de três diferentes dimensões: número de desenvolvedores, número de *commits* ou número de arquivos. Da seleção resultante, foram desconsiderados os sistemas com evidências de migração incorreta para o GitHub (a partir de outro repositório de código, tal como SVN). Para tanto, foram descartados sistemas nos quais mais de 50% de seus arquivos foram adicionados em menos de 20 *commits*. Isso foi realizado porque essa situação é considerada uma evidência de que boa parte do software foi desenvolvido utilizando outro sistema de controle de versão e que a migração para o GitHub não foi realizada de forma a preservar o histórico de desenvolvimento anterior.

A Tabela I caracteriza o *dataset* usado neste estudo (e proposto inicialmente, e descrito com mais detalhes, em [6]). Ele inclui 133 sistemas, desenvolvidos em seis linguagens diferentes; Ruby é a linguagem com mais sistemas (33) e PHP é a que possui menos sistemas (17). Considerando todos os sistemas, o *dataset* inclui mais de 63 mil desenvolvedores, 373 mil arquivos e 2 milhões de *commits*.

TABELA I
Dataset

Linguagem	Sistemas	Devs	Commits	Arquivos
JavaScript	22	5,740	108,080	24,688
Python	22	8,627	276,174	35,315
Ruby	33	19,960	307,603	33,556
C/C++	18	21,039	847,867	107,464
Java	21	4,499	418,003	140,871
PHP	17	3,329	125,626	31,221
Total	**133**	**63,194**	**2,083,353**	**373,115**

A Figura 1 apresenta a distribuição do número de desenvolvedores (Figura 1a), *commits* (Figura 1b) e arquivos (Figura 1c) em cada sistema do dataset. Para número de desenvolvedores, o primeiro, segundo e terceiro quartis são 111, 197 e 437, respectivamente. Os sistemas com mais desenvolvedores são torvalds/linux (15,8 mil), Homebrew/homebrew (5,6 mil) e rails/rails (3,2 mil desenvolvedores). Com relação ao número de commits, o primeiro, segundo e terceiro quartis são 1,950, 4,210 e 10,800, respectivamente. Os três sistemas com mais *commits* são torvalds/linux (505.6 mil), android/platform_frameworks_base (159.3 mil) e JetBrains/intellij-community (148.5 mil commits). Por fim, para número de arquivos dos sistemas, o primeiro, segundo e terceiro quartis são 269, 574 e 1.750, respectivamente. Os três sistemas com mais arquivos são JetBrains/intellij-community (74 mil), torvalds/linux (49 mil) e android/platform_frameworks_base (27 mil arquivos).



(a) Desenvolvedores (b) *Commits* (c) Arquivos

Fig. 1. Dataset

B. Descrição do Experimento

O presente estudo tem por objetivo comparar dois algoritmos para cálculo da métrica *truck factor*. A comparação será realizada sob dois aspectos: o tipo de abordagem utilizada para o cálculo da métrica e o tipo de abordagem utilizada para determinar autores de um sistema. Neste estudo, cada um desses aspectos é composto por dois fatores. O primeiro aspecto tem como fatores os algoritmos ZWK e AVL. O segundo aspecto, tipo de detecção de autoria, tem como fatores a autoria por *commit* e a autoria dada pelo *degree-of-authorship*, utilizados originalmente nos algoritmos Algoritmos 1 e Algoritmo 2, respectivamente. Tempo de execução e resultados quantitativos para *truck factor* serão as métricas utilizadas como parâmetro de comparação desses fatores. As questões de pesquisa definidas neste estudo são descritas a seguir.

(Q1) Qual o impacto do método de cálculo de autoria (autoria por *commit* ou autoria por DOA) no tempo de execução dos algoritmos ZWK e AVL?

O método utilizado para de detecção de autoria impacta diretamente no número de autores identificados no sistema em análise. Quanto mais abrangente é a abordagem, mais desenvolvedores são identificados como autores. Essa pergunta

[1] O GitHub permite que usuários informem o interesse em um determinado projeto marcando-o com uma estrela.

(Q1) tem por objetivo quantificar o impacto que o tipo de abordagem utilizada para identificação de autores no sistema tem no tempo de execução dos algoritmos. Para isso, serão tomadas duas implementações do algoritmo ZWK, uma que utiliza autoria por *commit* e outra que utiliza autoria por DOA. Os tempos de execução dessas abordagens serão comparados entre si. O mesmo será realizado com o algoritmo AVL.

(Q2) Qual o impacto do método de cálculo de autoria nos valores de *truck factor* produzidos pelos algoritmos ZWK e AVL?

A abordagem de detecção de autoria por *commit* aumenta o número de autores do sistema se comparada ao número de autores identificados, para o mesmo sistema, pela fórmula de DOA. A pergunta Q2 tem por objetivo identificar se o aumento do número de autores indica o aumento do *truck factor* de um sistema.

(Q3) Os tempos de execução dos algoritmos para cálculo de *truck factor* são significativamente diferentes?

O algoritmo ZWK é baseado em uma análise combinatória dos dados de autoria do sistema. Já o algoritmo AVL utiliza uma heurística gulosa para cálculo da métrica *truck factor*. Na pergunta Q3, tem-se por objetivo verificar se as duas abordagens são significativamente diferentes no que diz respeito ao tempo de execução. Além disso, busca-se identificar qual algoritmo é possui desempenho estatisticamente melhor. Para responder essa questão, o tempo de execução do código descrito no Algoritmo 1 e Algoritmo 2 será medido.

Algoritmos que fazem uso de análise combinatórias são, por natureza, algoritmos de força bruta. Dessa forma, a hipótese que se investiga é se os algoritmos são significativamente diferentes e que se AVL possui melhor desempenho em relação ao ZWK no que diz respeito ao tempo de execução.

(Q4) Existem diferenças nos resultados de *truck factor* produzidos pelos algoritmos ZWK e AVL?

Na questão de pesquisa Q4, busca-se analisar as possíveis diferenças entre os resultados apresentados pelos algoritmos ZWK e AVL. Será realizada um análise quantitativa dos resultados dos algoritmos, ou seja, serão considerados os valores apresentados para a métrica *truck factor*.

Para responder às questões de pesquisa, cada sistema do *dataset* foi representado por dois arquivos: *sistema_doa.csv* e *sistema_commit.csv*. Cada arquivo representa os dados do sistema (arquivo e autor) de acordo com uma abordagem de detecção de autoria. Os arquivos *sistema_doa.csv* representam os dados de autoria obtidos através da aplicação da fórmula DOA, utilizada pelos autores do Algoritmo 2. O padrão *sistema_commit.csv* indica arquivos cujo conteúdo representa os dados obtidos por meio da abordagem autoria por *commit* utilizada na versão original do Algoritmo 1.

Uma vez que a codificação original do algoritmo ZWK não está publicamente disponível, houve a necessidade de sua implementação neste trabalho. Para realizar uma comparação justa, ambos os algoritmos foram implementados neste trabalho tendo como base os pseudocódigos apresentados nos trabalhos descritos na Seção II. Ambos foram desenvolvidos na linguagem *Java*, uma vez que foi essa a linguagem utilizada na implementação original dos algoritmos. Como descrito no Algoritmo 2, o limiar de cobertura da heurística é um valor fixo que representa 50% dos arquivos do sistema. Sendo assim, o mesmo limiar (50%) foi utilizado como valor para o parâmetro *cobertura* para a heurística ZWK. As listas *devs*, *arquivos* e o *Mapeamento "A"* foram preenchidas com dados contidos nos arquivos *sistema_doa.csv* e *sistema_commit.csv*.

Os experimentos aqui descritos foram realizados em um computador *desktop* com as seguintes configurações: processador Intel(R) Core(TM) i5-3350P CPU @ 3.10GHz; 8GB de RAM e sistema operacional Windows 10.

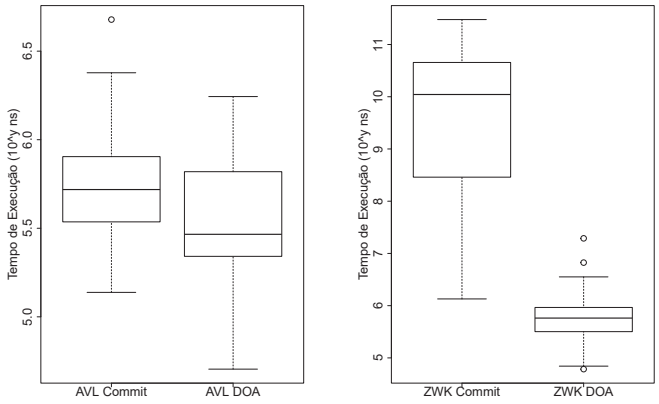
IV. Resultados

Esta seção apresenta os resultados obtidos nos estudos experimentais realizados para cada questão de pesquisa definida neste trabalho. Foram analisados dados de 133 softwares. No caso do algoritmos AVL, foram considerados dados de todos esses sistemas, tanto no caso em que se utiliza a cálculo de autoria via *commit* como no caso de autoria dada por DOA. No caso do algoritmo ZWK, por se tratar de algoritmo cuja abordagem é *força bruta*, não foi possível a obtenção de dados de todos os software; na implementação em que se utiliza a autoria via *commit*, foram obtidos dados de 13 softwares, e na implementação em que se utiliza autoria dada por DOA, foram obtidos dados de 75 softwares. Os resultados são descritos a seguir.

(Q1) Qual o impacto do método de cálculo de autoria (autoria por commit ou autoria por DOA) no tempo de execução dos algoritmos ZWK e AVL?

Para responder a esta questão de pesquisa, é necessário comparar os tempos de execução de um algoritmo quando são utilizadas formas de cálculo de autoria diferentes. A unidade de tempo de execução considerada para os algoritmos é nanosegundo. O gráficos das Figura 2 mostram a dispersão dos tempos de execução obtidos para os algoritmos, para cada abordagem de autoria. Para melhor visualização dos dados, os gráficos estão em escala logarítmica.

Observa-se que o tempo de execução do algoritmo AVL, quando se utiliza a autoria por DOA é levemente menor do que quando se utiliza a autoria por *commit*. O intervalo dos valores dos tempos de execução obtidos são próximos. No caso da autoria por *commit*, o tempo de execução está no intervalo $[10^5;10^7]$, com mediana $10^{5,71}$ nanosegundos ($0,51$ milissegundos). No caso da autoria por DOA, o tempo de



(a) Algoritmo AVL (b) Algoritmo ZWL

Fig. 2. Tempo de execução por abordagem de autoria

execução está no intervalo $[10^4;10^6]$, com mediana $10^{5,47}$ nanosegundos ($0,30$ milissegundos).

Os tempos de execução para ambas as abordagens não possuem distribuição normal. Sendo assim, para identificar qual abordagem proporciona menor tempo de execução do algoritmo, utilizaremos o teste de *Wilcoxon* para amostras pareadas. Os resultados da comparação entre DOA e *commit* mostram que é possível afirmar, com 95% de confiança(p-value = $0,0171$), que o tempo de execução da implementação do algoritmo AVL que utiliza DOA é menor do que aquela que utiliza *commit*.

A mesma análise foi realizada para o algoritmo ZWK. Nesse caso, observa-se que o tempo de execução quando se utiliza a autoria por DOA é muito menor do que quando se utiliza a autoria por *commit*. Além disso, a variação do tempo de execução para os softwares analisados é maior quando se utiliza a autoria por *commit*. No caso da autoria por *commit*, o tempo de execução está no intervalo $[10^6;10^{11}]$, com mediana 10^{10} nanosegundos ($\approx$ 11 segundos). No caso da autoria por DOA, o tempo de execução está no intervalo $[10^5;10^7]$, com mediana $10^{5,76}$ nanosegundos (0,58 milissegundos).

Os resultados da comparação entre DOA e *commit* para o algoritmo ZWK indicam que, com 95% de confiança (p-value = $0,0001$), pode-se afirmar que o tempo de execução da implementação do algoritmo ZWK que utiliza DOA é menor do que aquela que utiliza *commit*.

(Q2) Qual o impacto do método de cálculo de autoria nos valores de truck factor produzidos pelos algoritmos ZWK e AVL?

Nesta questão de pesquisa, investiga-se se o método de cálculo de autoria implica em geração de resultados diferentes para a métrica de *truck factor*, para um mesmo algoritmo, ZWK ou AVL.

As Figuras 4 e 3 mostram os resultados da métrica *truck factor* gerada para os 133 sistemas da amostra pelo algoritmo AVL, utilizando-se a autoria por *commit* e DOA. Esses resultados mostram que na maior parte dos casos, 88%, o valor *truck factor* gerado na implementação que utiliza DOA é menor. Isso indica que a forma de autoria empregada no algoritmo AVL tem impacto importante no valor do *truck factor* resultante. Em média, a diferença dos resultados gerados pela implementação que utiliza *commit* para a que utiliza DOA é $30,2$. O intervalo de confiança da média, a 95%, é [119,8; 130,1]. A média obtida na amostra está fora desse intervalo; porém, a amostra conta com um *outlier*, o Linux. No Linux, a diferença observada foi de 610: a implementação com *commit* gera *truck factor* igual a 57, e a com DOA, 667. Dessa forma, pode-se afirmar que o valor *truck factor* gerado na implementação que utiliza DOA é de fato menor, no caso do algoritmo AVL.

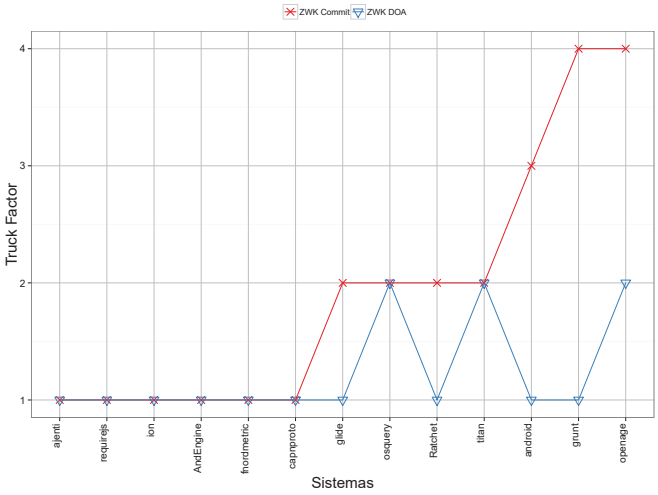


Fig. 5. Truck factor ZWL vs. AVL - autoria por DOA

Os resultados da métrica *truck factor* gerados pelo algoritmo ZWK são mostrados na Figura 5. Para esse algoritmo, foram coletados dados de 13 sistemas. Nesse caso, em 38% dos sistemas o valor do *truck factor* gerado na implementação que utiliza DOA é menor. Na maior parte dos casos, 62%, os resultados do *truck factor* são iguais. Nota-se que esse resultado observado nos 13 sistemas analisados no caso do algoritmo ZWK é o mesmo do algoritmo AVL, ou seja, no caso desse sistema especificamente, o algoritmo AVL se comportou da mesma maneira. Em média, a diferença dos resultados gerados pela implementação que utiliza *commit* para a que utiliza DOA é $0,75$. O intervalo de confiança da média, a 95%, é [0,64; 1,46]. Dessa forma, também no caso do algoritmo ZWK, pode-se afirmar que o valor *truck factor* gerado na implementação que utiliza DOA é menor.

(Q3) Os tempos de execução dos algoritmos para cálculo de truck factor são significativamente diferentes?

Com o propósito de responder esta questão de pesquisa, os tempos de execução dos algoritmos ZWK e AVL foram comparados de duas formas: compararam-se os tempos obtidos das execuções desses algoritmos utilizando-se a autoria por *commit*

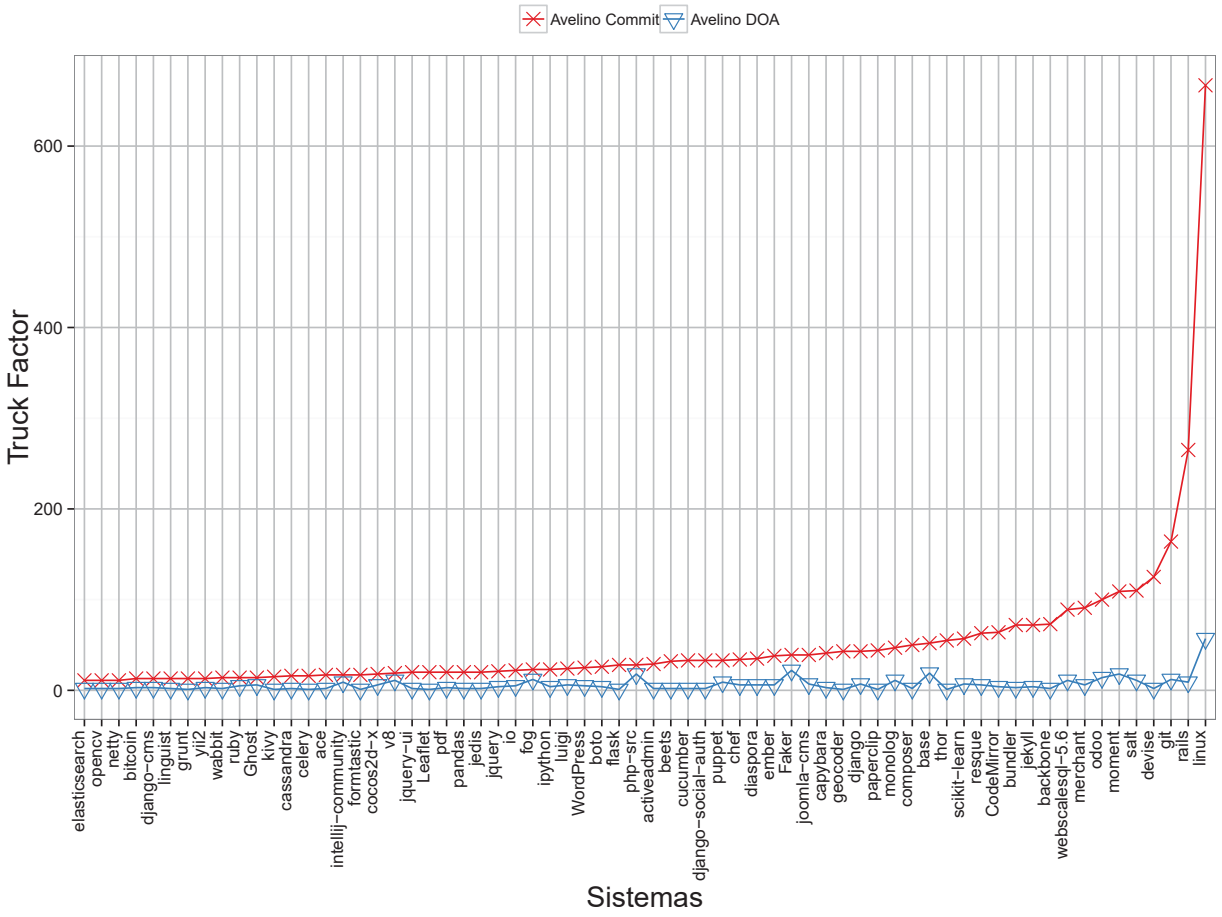
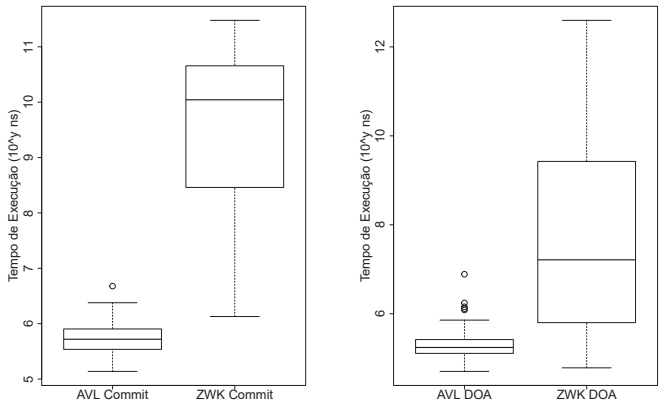


Fig. 3. Truck Factor AVL

e também compararam-se os tempos obtidos das execuções desses algoritmos utilizando-se a autoria por DOA. Os gráficos das Figuras 6(a) e 6(b) mostram essas duas comparações, respectivamente.



(a) Commit (b) DOA

Fig. 6. Tempo de execução dos algoritmos

Graficamente, nota-se que o tempo de execução do algoritmo AVL é muito menor do que ZWK. Observa-se, também, que a distribuição dos valores de tempos de execução de AVL é mais simétrica do que a de ZWK. O tempo de execução do algoritmo AVL utilizando *commit* está no intervalo $[10^5;10^6]$, com mediana 10^5, enquanto o do algoritmo ZWK está no intervalo $[10^6;10^{11}]$, com mediana 10^{10}.

A diferença média entre os tempos de execução entre os algoritmos ZWK e AVL é de 10^{10} nanosegundos. O teste de *Wilcoxon* para amostras pareadas, foi realizado para identificar a significância dessa diferença. Os resultados obtidos indicam que pode-se afirmar com 99% de confiança (p-value=0,0002) que o tempo de execução da implementação do algoritmo AVL é significativamente menor do que o do algoritmo ZWK, quando se utiliza autoria por *commit*.

Conclusão similar é obtida ao analisar os dados da comparação entre os tempos de execução dos algoritmos quando se utiliza autoria por DOA. Nesse caso, a diferença média entre os tempos de execução entre os algoritmos ZWK e AVL é de 10^{11} nanosegundos. Os resultados obtidos mostram que, pode-se afirmar que o tempo de execução da

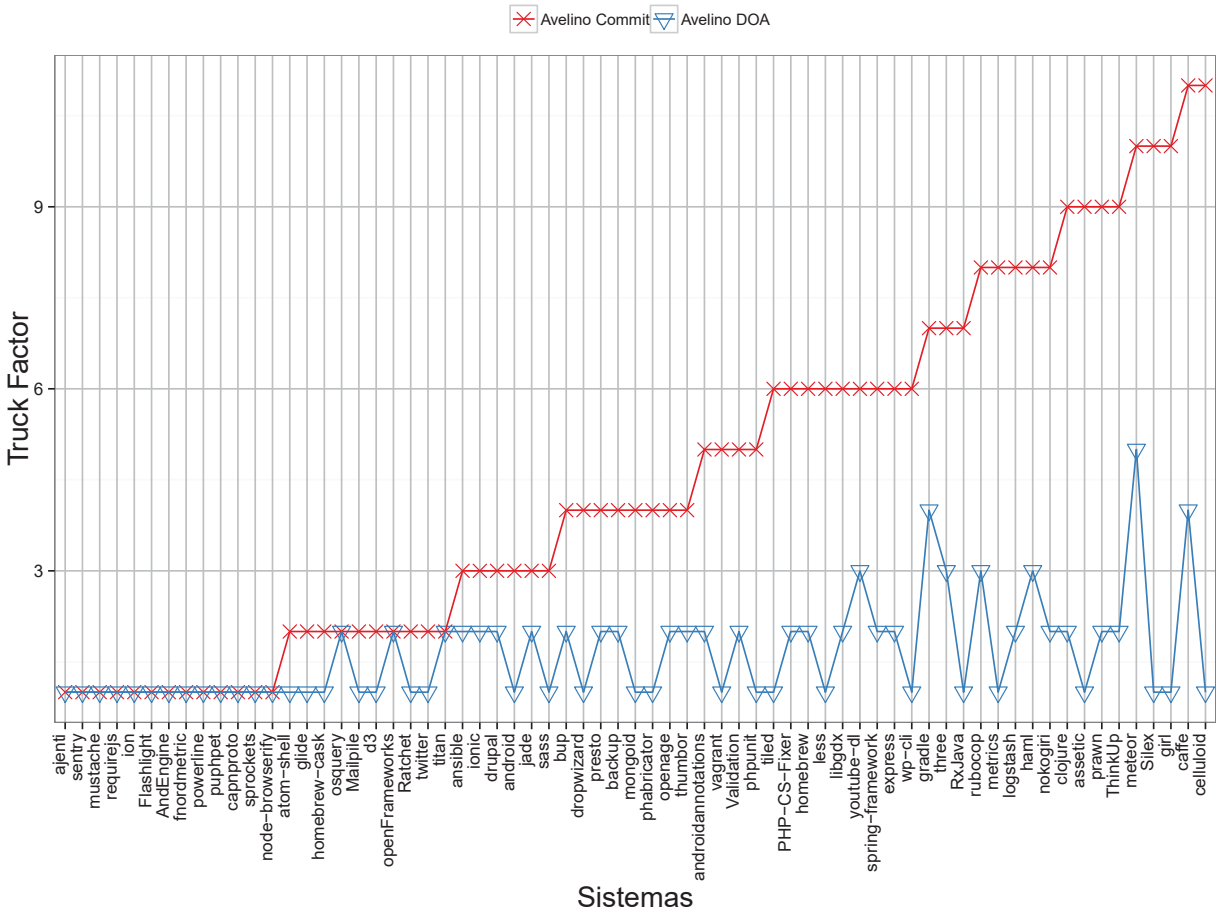


Fig. 4. Truck Factor AVL

implementação do algoritmo AVL é significativamente menor do que o do algoritmo ZWK, quando se utiliza autoria por DOA, tendo em vista o nível de confiança obtido nos testes: 99% de confiança (p-value=0,0001).

(Q4) Existem diferenças nos resultados de truck factor produzidos pelos algoritmos ZWK e AVL?

Para responder esta questão de pesquisa, os resultados da métrica *truck factor* dos algoritmos ZWK e AVL foram comparados de duas formas: a primeira utilizando-se a autoria por *commit* e a segunda utilizando-se a autoria por DOA.

Como a execução do algoritmo ZWK foi possível para 75 sistemas utilizando-se o cálculo de autoria por DOA, e para 13 sistemas utilizando o cálculo de autoria por *commit*, para efeitos de comparação entre os algoritmos ZWK e AVL, os gráficos das Figuras 7 e 8 mostram dados da comparação de 75 e 13 sistemas, respectivamente.

A Figura 7 mostra os dados de *truck factor* gerados pelos algoritmos empregando autoria por DOA. Observa-se que os resultados de ambos os algoritmos são idênticos, exceto em dois sistemas: *jnicklas* e *spotify*. A Figura 8 mostra os dados de *truck factor* gerados pelos algoritmos empregando autoria por *commit*. Também nesse caso, os resultados de ambos os algoritmos são idênticos, exceto para o sistema *grunt*.

V. Discussão

Este trabalho analisa dois algoritmos para cálculo de *truck factor* propostos na literatura. O objetivo deste estudo comparativo é analisar os comportamentos desses algoritmos quanto ao tempo de execução e quanto ao resultados gerados. Para tal, foram realizadas quatro tipos de análises: (1) verificar o impacto do método de autoria no tempo de execução dos algoritmos, (2) verificar o impacto do método de autoria no valor do *truck factor* obtidos pelos algoritmos, (3) comparar a diferença entre o desempenho dos algoritmos quanto ao tempo de execução e (4)comparar os algoritmos quanto aos valores de *truck factor* gerados.

A amostra utilizada no estudo é relevante. Foram utilizados dados de 133 sistemas abertos, desenvolvidos em linguagens de programação populares. Os sistemas selecionados para análise são também populares. Dentre os sistemas analisados, estão alguns reconhecidos pelo grande volume de contribuidores, como é o caso do Linux. Essas características são

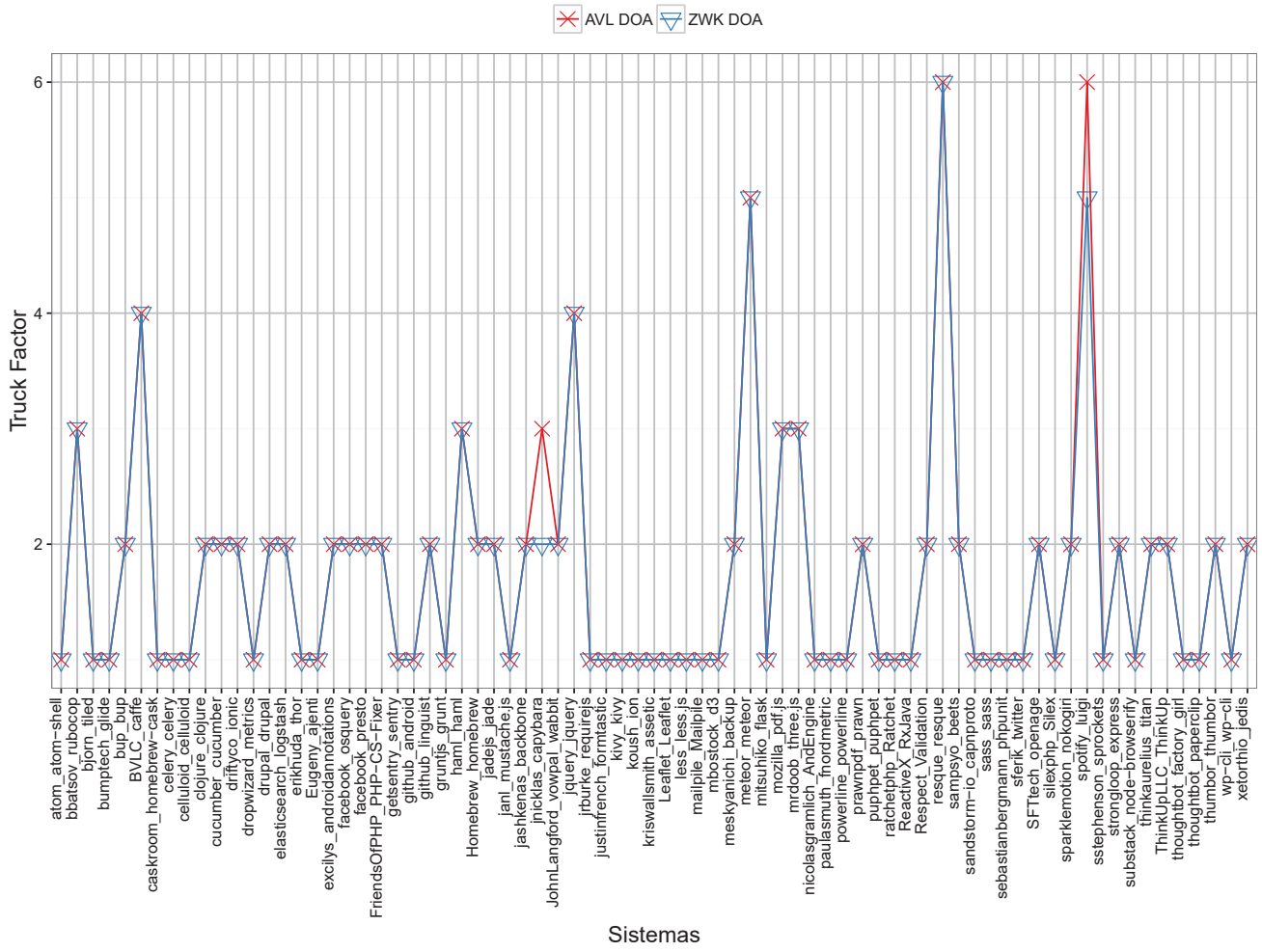


Fig. 7. Truck factor ZWL vs. AVL - autoria por DOA

importantes para as análises realizadas neste trabalho, uma vez que a objetivo dos algoritmos estudados está relacionada ao conhecimento do software pelos seus desenvolvedores.

Os resultados obtidos neste estudo revelam algumas conclusões importantes a respeito dos dois algoritmos de cálculo de *truck factor* analisados. A primeira delas é que o método de autoria tem impacto no tempo de execução de ambos os algoritmos. Embora esse resultado não seja surpreendente, este estudo provê evidências empíricas a respeito da premissa de que a utilização da autoria por meio de DOA reduz o tempo para computar o *truck factor*. Isso ocorre porque o DOA considera uma quantidade menor de autores em sua análise.

Pelo estudo realizado, conclui-se que o método de autoria também tem impacto no valor do *truck factor* gerado pelos algoritmos. Os valores de *truck factor* obtidos quando se utiliza DOA é menor do que quando se utiliza autoria por *commit*. Também, nesse caso, a explicação para esse resultado é a redução de autores realizada pelo DOA. Há, porém, casos em que os valores de *truck factor* são iguais, independente do método de autoria empregado. São casos em que o *truck factor* é pequeno, ou seja, há grande concentração de conhecimento do sistema em uma quantidade pequena de desenvolvedores.

Sabe-se que o cálculo de *truck factor* é um problema NP-Difícil. No entanto, não se sabe até quando soluções força bruta são viáveis. Os resultados apresentados neste trabalho revelam que uma solução força bruta, tal como adotada no algoritmo ZWK, escala até sistemas com cerca de 33 desenvolvedores. Adicionalmente, não se conhece também os ganhos de escalabilidade proporcionados por soluções heurísticas. Este trabalho mostra que uma solução heurística, tal como proposta no algoritmo AVL, escala até projetos com pelo menos 11.319 desenvolvedores, tal como o projeto *Linux*, presente no *dataset* utilizado. Dessa forma, o estudo empírico é importante para se observar como as soluções propostas para o cálculo de *truck factor* se comportam na prática, com dados obtidos de cenários reais da Engenharia de Software.

Entre os dois algoritmos analisados, o AVL apresenta melhor desempenho quanto ao tempo de execução resposta. Esse resultado é esperado pelo fato de o algoritmo ZWK ser do tipo *força bruta*, ou seja, tem complexidade exponencial. Neste

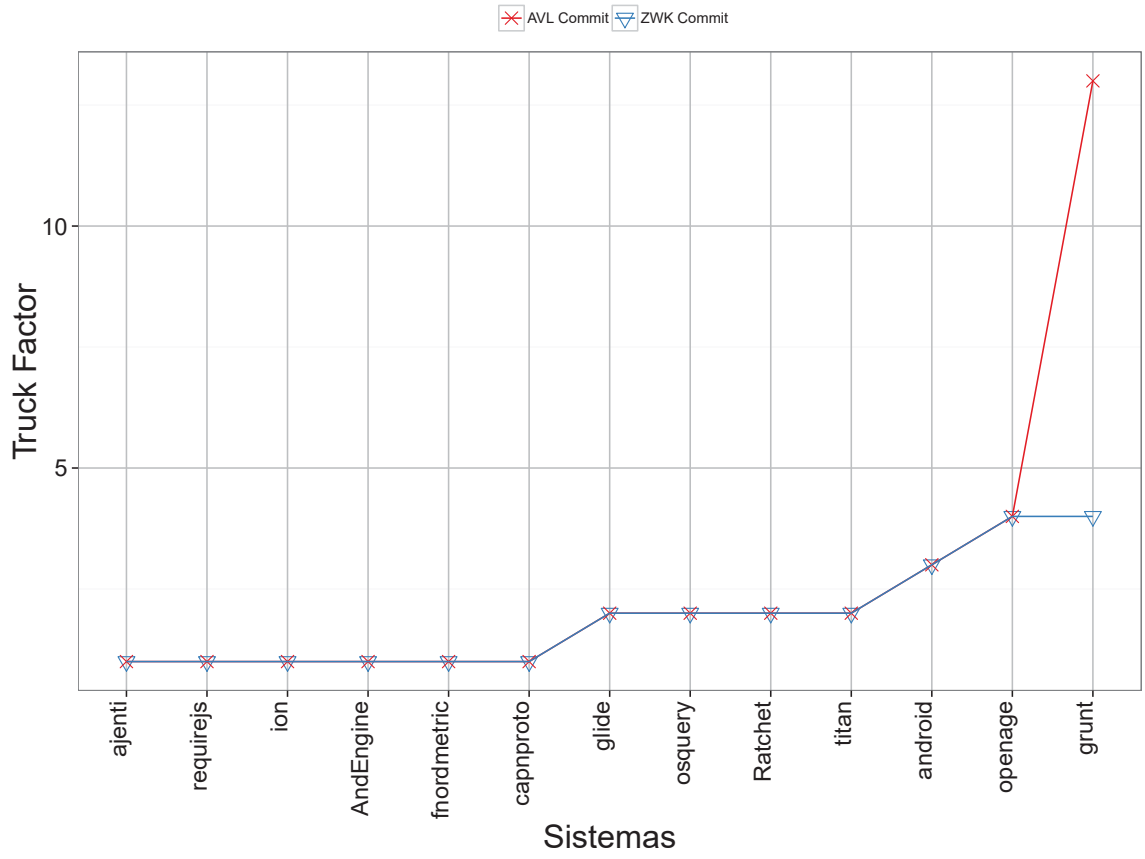


Fig. 8. Truck factor ZWL vs. AVL - autoria por Commit

estudo, porém, evidenciou-se quão melhor é o algoritmo AVL em comparação a ZWK quanto ao tempo de execução. A diferença média do tempo de execução do ZWK para o AVL é de 10^{10} nanosegundos (10 segundos) quando se utiliza *commit*, e de 10^{11} nanosegundos (1,7 minutos) quando se utiliza DOA. Além disso,

O dois algoritmos analisados apresentam, em geral, em resultados de *truck factor* iguais. Esse comportamento é observado tanto no caso das implementações que utilizam autoria por *commit* quanto nas implementações que utilizam DOA. O algoritmo ZWK analisa todas as combinações possíveis de autores para gerar o *truck factor*. A partir dos resultados obtidos, conclui-se que o algoritmo AVL obtém os mesmos resultados, porém em um tempo menor.

O *truck factor* é uma métrica que visa informar a quantidade de desenvolvedores que podem ser eliminados da equipe de forma a manter o conhecimento do sistema com ela. Quanto a isso, os resultados obtidos não permitem asserções acerca da correção da forma do cálculo do *truck factor*. Para que tal correção fosse averiguada, seria necessário, por exemplo, realizar um estudo empírico envolvendo a opinião dos desenvolvedores dos sistemas. Essa investigação, embora importante, não é um dos objetivos do presente trabalho.

VI. Ameaças à Validade

Validade Externa Neste trabalho foram utilizados 133 projetos para realização da comparação. Embora seja uma amostra considerável, não podemos afirmar que os algoritmos apresentados neste trabalho apresentarão o mesmo comportamento para qualquer projeto diferente dos projetos contidos no *dataset*. Para contornar essa questão, foram utilizados os projetos de software mais populares do GitHub, distribuídos em seis linguagens de programação: *JavaScript*, *Python*, *Ruby*,*C/C++*,*Java* e *PHP*. Além disso há uma grande variedade de projetos em relação, ao número de arquivos de código-fonte e número de desenvolvedores, fatores estes que afetam diretamente o desempenho dos algoritmos estudados.

Validade Interna Neste estudo foram coletados dados referentes ao tempo de execução dos algoritmos ZWK e AVL. Para isso foi avaliado o tempo de execução do trecho de código indicado nos Algoritmos1 e 2. Uma vez que contamos apenas com a re-implementação dos algoritmos, não é possível afirmar que as mesma estruturas de dados foram utilizadas na codificação original. Buscando minimizar o impacto dessa questão nos resultados, as duas implementações utilizadas neste trabalho fizeram uso das mesmas estruturadas de dados. Além disso, buscou-se utilizar estruturadas de dados

otimizadas a fim de minimizar o impacto delas no tempo de execução do algoritmo. O tempo de execução do cálculo de DOA, não foi considerado no tempo de execução dos algoritmos. Entretanto, considerou-se que esse cálculo possui impacto pequeno no tempo de execução uma vez que trata-se de um algoritmo com tempo linear (*O(n)*).

VII. Trabalhos Relacionados

Apesar de não ser um conceito novo, existem poucos trabalhos que exploram *truck factor* como uma métrica para identificar concentração de conhecimento em ambientes de desenvolvimento de *software*. No melhor de nosso conhecimento, apenas três trabalhos propõem uma formalização dos passos para calcular o *truck factor* de um sistema. Além dos já mencionados—ZWK [2] e AVL [6]—uma terceira abordagem é apresentada por Cosentino et al. [10]. Os autores desse trabalho propõe uma abordagem hierárquica, na qual o conhecimento sobre o código é inicialmente calculado para os arquivos do sistema, sendo então combinado para determinar autoria em nível de módulos e posteriormente em nível do sistema. Apenas autores com um determinado nível de conhecimento sobre a unidade do sistema são considerados no cálculo do *truck factor*. A abordagem requer a configuração de um conjunto de parâmetros e a seleção de métricas de conhecimento de código, o que dificulta sua utilização em estudos comparativos e dessa forma não foi avaliada nesse trabalho.

Tendo como base o algoritmo ZWK, outros trabalhos propuseram o uso do *truck factor* como métrica para identificar "heróis" em sistemas de código aberto [4]. Existem ainda estudos que visam identificar valores de referência para uso do truck factor em sistemas reais [11], ou ainda, exploram dificuldades do cálculo [9] e a complexidade computacional do algoritmo [5].

Em resumo, os trabalhos anteriores contribuem com o estudo do *truck factor* propondo algoritmos, avaliações e aplicações desse, sem entretanto comparar seus resultados com estudos prévios. Dessa forma, nosso trabalho expande o conhecimento na área ao apresentar os resultados de um estudo comparativo entre dois algoritmos para calculo de *truck factor*, analisando tanto o desempenho dos algoritmos como os valores inferidos.

VIII. Conclusão

Truck factor é uma métrica importante, pois permite revelar a existência de ilhas de conhecimento em equipes de desenvolvimento de software. Apesar disso, existem poucos algoritmos para cálculo dessa métrica e nenhuma comparação entre tais algoritmos foi até hoje realizada, no melhor de nosso conhecimento. Os resultados do trabalho, então, trazem conhecimento sobre as diferenças entre os dois principais algoritmos para cálculo de *truck factor* propostos até o momento. Tais diferenças foram observadas empiricamente, com dados da prática de software. Sendo assim, os resultados do trabalho permite, aos gerentes de projeto de software, escolher melhor o algoritmo que aplicarão para o cálculo de *truck factor* em suas equipes.

Este trabalho teve por objetivo realizar um estudo comparativo de dois algoritmos proposto na literatura para o cálculo de *truck factor*. Foram conduzidos experimentos com objetivo de identificar como o tipo de autoria utilizado pode afetar o comportamento de um mesmo algoritmo em relação a dois aspectos: tempo de execução e resultados gerados pelos algoritmos. Além disso, buscou-se comparar os algoritmos entre si em função desses dois aspectos.

Os resultados obtidos indicam, com 95% de confiança, que autoria por *commit* é a abordagem que possui maior impacto sobre os algoritmos, tanto em relação aos resultados gerados quanto em relação ao tempo de execução. Em relação aos algoritmos, os resultados apontam, com 95% de confiança que o tempo de execução apresentando pelo algoritmo ZWK é sempre superior ao tempo apresentado por AVL. Em relação aos valores de *truck factor* apresentados, concluiu-se que, de forma geral, os algoritmos apresentam resultados iguais.

Como trabalhos futuros são propostos: a investigação qualitativa dos dados, para identificar possíveis divergências entre o conjunto de desenvolvedores indicados como *truck factor* nos algoritmos; a validação dos resultados gerados pelos algoritmos, realizada por pessoas ligadas ao desenvolvimento dos softwares avaliados.

Agradecimentos

Esta pesquisa foi financiada pela FAPEMIG, CAPES e pelo CNPq.

Referências

[1] B. Boehm, "Software risk management: principles and practices," *IEEE Software*, vol. 8, no. 1, pp. 32–41, 1991.

[2] N. Zazworka, K. Stapel, E. Knauss, F. Shull, V. R. Basili, and K. Schneider, "Are developers complying with the process: an xp study," in *4th International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2010, pp. 14:1–14:10.

[3] M. Bowler, "Truck factor," 2005. [Online]. Available: http://www.agileadvice.com/2005/05/15/agilemanagement/truck-factor/

[4] F. Ricca and A. Marchetto, "Are heroes common in FLOSS projects?" in *4th International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2010, pp. 1–4.

[5] C. Hannebauer and V. Gruhn, "Algorithmic complexity of the truck factor calculation," in *Product-Focused Software Process Improvement*. Springer, 2014, pp. 119–133.

[6] G. Avelino, L. Passos, A. Hora, and M. T. Valente, "A novel approach for estimating truck factors," in *24th International Conference on Program Comprehension (ICPC)*, 2016, pp. 1–10.

[7] T. Fritz, J. Ou, G. C. Murphy, and M.-E. Hill, "A degree-of-knowledge model to capture source code familiarity," in *32nd International Conference on Software Engineering (ICSE)*, 2010, pp. 385–394.

[8] T. Fritz, G. C. Murphy, M.-E. Hill, J. Ou, and E. Hill, "Degree-of-knowledge: modeling a developer's knowledge of code," *Software Engineering and Methodology*, vol. 23, no. 2, pp. 14:1–14:42, 2014.

[9] F. Ricca, A. Marchetto, and M. Torchiano, "On the difficulty of computing the truck factor," in *12th International Conference on Product-focused Software Process Improvement (PROFES)*, 2011, pp. 337–351.

[10] V. Cosentino, J. Izquierdo, and J. Cabot, "Assessing the bus factor of git repositories," in *22nd International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2015, pp. 499–503.

[11] M. Torchiano, F. Ricca, and A. Marchetto, "Is my project's truck factor low?: theoretical and empirical considerations about the truck factor threshold," in *2nd International Workshop on Emerging Trends in Software Metrics (WETSoM)*, 2011, pp. 12–18.