Tweaking Association Rules to Optimize Software Change Recommendations

Mairieli Santos Wessel
Universidade Tecnológica Federal do Paraná (UTFPR)
Campo Mourão, Paraná, Brasil
mairieliw@alunos.utfpr.edu.br

Maurício Finavaro Aniche
Delft University of Technology
Delft, South Holland, The Netherlands
m.f.aniche@tudelft.nl

Gustavo Ansaldi Oliva
Queen's University
Kingston, Ontario, Canada
gustavo@cs.queensu.ca

Marco Aurélio Gerosa
Northern Arizona University
Flagstaff, Arizona, United States
marco.gerosa@nau.edu

Igor Scaliante Wiese
Universidade Tecnológica Federal do Paraná (UTFPR)
Campo Mourão, Paraná, Brasil
igor@utfpr.edu.br

ABSTRACT

Past researchs have been trying to recommend artifacts that are likely to change together in a task to assist developers in making changes to a software system, often using techniques like association rules. Association rules learning is a data mining technique that has been frequently used to discover evolutionary couplings. These couplings constitute a fundamental piece of modern change prediction techniques. However, using association rules to detect evolutionary coupling requires a number of configuration parameters, such as measures of interest (e.g. support and confidence), their cut-off values, and the portion of the commit history from which co-change relationships will be extracted. To accomplish this set up, researchers have to carry out empirical studies for each project, testing a few variations of the parameters before choosing a configuration. This makes it difficult to use association rules in practice, since developers would need to perform experiments before applying the technique and would end up choosing non-optimal solutions that lead to wrong predictions. In this paper, we propose a fitness function for a Genetic Algorithm that optimizes the co-change recommendations and evaluate it on five open source projects (CPython, Django, Laravel, Shiny and Gson). The results indicate that our genetic algorithm is able to find optimized cut-off values for support and confidence, as well as to determine which length of commit history yields the best recommendations. We also find that, for projects with less commit history (5k commits), our approach produced better results than the regression function proposed in the literature. This result is particularly encouraging, because repositories such as GitHub host many young projects. Our results can be used by researchers when conducting co-change prediction studies and by tool developers to produce automated support to be used by practitioners.

SBES'17, Fortaleza, CE, Brazil

DOI: 10.1145/3131151.3131163

CCS CONCEPTS

•**Software and its engineering** →**Software evolution;** *Maintaining software;* •**Information systems** →**Recommender systems;** *Association rules;*

KEYWORDS

Change Recommendation, Association Rules, Genetic Algorithm

ACM Reference format:
Mairieli Santos Wessel, Maurício Finavaro Aniche, Gustavo Ansaldi Oliva, Marco Aurélio Gerosa, and Igor Scaliante Wiese. 2017. Tweaking Association Rules to Optimize
Software Change Recommendations. In *Proceedings of SBES'17, Fortaleza, CE, Brazil, September 20–22, 2017,* 10 pages.
DOI: 10.1145/3131151.3131163

1 INTRODUÇÃO

De acordo com as leis de Lehman [15], à medida que um sistema evolui, sua estrutura torna-se mais complexa. A complexidade afeta diretamente como o sistema é modificado. Particularmente, essa complexidade afeta em grande parte desenvolvedores novatos e contribuidores casuais de projetos de software livre, uma vez que a falta de conhecimento sobre a arquitetura do software cria dificuldades em encontrar quais artefatos devem ser modificados para resolver uma determinada tarefa [21, 26].

Para auxiliar os desenvolvedores, foram propostas diversas técnicas de *recomendação de mudanças* [3, 7, 12, 23, 29]. Várias dessas técnicas se baseiam no algoritmo de *regras de associação*, cuja premissa é a de que artefatos que mudam juntos frequentemente no passado são propensos a mudar juntos no futuro. Mais especificamente, o resultado da execução desse algoritmo é uma lista de *acoplamentos de mudança* (também chamados de *acoplamentos evolucionários*) [20]. Diz-se que há um acoplamento de mudança de um artefato A para outro artefato B quando as mudanças em B frequentemente implicam em uma mudança em A. As técnicas baseadas em regras de associação se valem desses acoplamentos para produzirem suas recomendações.

Entretanto, a acurácia das recomendações produzidas por técnicas baseadas em regras de associação é afetada por um conjunto de fatores, tais como: tamanho do histórico de mudanças

do projeto (também chamado de conjunto de treinamento), configuração de medidas de interesse (por exemplo, suporte e confiança) usadas para filtrar os acoplamentos de mudança e pela inclusão de novas mudanças (*commits*) que ocorrem durante a evolução do software. Embora os guias práticos atuais tenham investigado diversos fatores que influenciam a acurácia das recomendações [18, 19], eles não definiram valores recomendados para configuração das medidas de interesse. Além disso, os valores ótimos variam para cada projeto.

O objetivo deste trabalho é investigar como determinar empiricamente limiares ótimos de suporte, confiança e o tamanho do conjunto de treinamento que geram as melhores recomendações de mudança. A abordagem proposta define uma função de aptidão (*fitness function*) para um algoritmo genético que visa encontrar o conjunto de treinamento com base no histórico de modificações (*commits*) e limiares de suporte e confiança que otimizam as recomendações de mudança para um projeto de software. Para avaliação da abordagem, comparamos a acurácia das recomendações produzidas pelo modelo proposto com um modelo estático gerado a partir de uma função de regressão proposta por Moonen et al. [19]. O modelo estático retorna o valor ideal para o tamanho do conjunto de treinamento; no entanto, não define valores ideais para os limiares de suporte e confiança. Para comparação das abordagens foram adotados limiares de suporte e confiança usados na literatura [2, 29].

Para avaliar o modelo proposto, foram utilizados dados de cinco projetos de código aberto (CPython, Django, Laravel, Shiny e Gson) para responder duas questões de pesquisa: **(QP1)** Como a acurácia do modelo de recomendação de mudanças baseado em Algoritmo Genético se compara àquela do modelo estático proposto por Moonen et al.[19]? **(QP2)** Como a acurácia dos modelos de recomendação de mudança se comporta quando o conjunto de teste aumenta?

Os resultados indicam que o modelo baseado em algoritmo genético apresenta uma acurácia similar à abordagem de Moonen et al. [19]. Entretanto, o modelo proposto se ajustou a diferentes tamanhos de projetos, enquanto que o modelo de regressão não conseguiu identificar o tamanho do conjunto de treinamento capaz de ser usado na prática para um dos projetos, uma vez que o tamanho do conjunto de treinamento recomendado foi maior do que a quantidade de histórico disponível do projeto. Esse resultado é particularmente interessante porque a maioria dos projetos hospedados em repositórios como o GitHub não têm histórico de mudanças grande [22].

Este trabalho está organizado da seguinte forma: a Seção 2 discute regras de associação; a Seção 3 apresenta o problema de pesquisa; a Seção 5 apresenta detalhes da abordagem de recomendação de mudanças baseada em algoritmo genético; a Seção 6 apresenta os resultados e discussão das avaliações realizadas; a Seção 4 apresenta os trabalhos relacionados; a Seção 7 apresenta as ameaças à validade do estudo e a Seção 8 apresenta as conclusões e planos para trabalhos futuros.

2 REGRAS DE ASSOCIAÇÃO

Uma regra de associação [1] é um tipo especial de implicação expressa por $X \Rightarrow Y$ e que é lida da seguinte forma "quando X ocorre, Y tende a ocorrer"(o contrário não é necessariamente verdade). Mais especificamente, X e Y são conjuntos disjuntos de itens, sendo X chamado de *antecedente* e Y chamado de *consequente*.

Duas medidas de interesse são calculadas para filtrar as regras relevantes: suporte e confiança. O suporte é a medida que representa a frequência de uma regra no conjunto de transações. Um conjunto de regras que aparece em muitas transações é dito ser frequente. O suporte da regra $X \Rightarrow Y$, escrito como $Suporte(X \Rightarrow Y)$, indica o número de transações que contêm tanto X quanto Y. A força de uma regra, por sua vez, é medida pela confiança. A confiança, escrita como $Confiança(X \Rightarrow Y)$, determina o quão frequente Y aparece em transações que contêm X.

Seja a frequência de um conjunto de itens X em um conjunto de transações θ definida como $freq_\theta(X) = |\{T \in \theta, X \subseteq T\}|$, as medidas de interesse $Suporte$ e $Confiança$ são definidas formalmente como:

$$Suporte(X \Rightarrow Y) = freq_\theta(X \cup Y)$$

$$Confiança(X \Rightarrow Y) = \frac{Suporte(X \Rightarrow Y)}{Suporte(X)} = \frac{freq_\theta(X \cup Y)}{freq_\theta(X)}$$

Assim, dado um conjunto de transações, a mineração por regras de associação gera um conjunto de regras que contenham suporte e confiança maiores ou iguais aos valores mínimos informados. O algoritmo *Apriori* [1] é frequentemente usado para se obter regras de associação eficientemente.

3 DESCRIÇÃO DO PROBLEMA

No contexto de desenvolvimento de software, os sistemas de controle de versão armazenam o histórico de mudanças dos artefatos. Logo, um *commit* pode ser interpretado como uma transação que contêm um conjunto de arquivos modificados. Assim, minerando o sistema de controle de versão, é possível extrair regras de associação como $\{x\} \Rightarrow \{y\}$, indicando que "quando um desenvolvedor modifica o artefato x, ele tende a também modificar o artefato y". Portanto, regras de associação são uma forma natural de se identificar e expressar *acoplamentos de mudança* entre artefatos de software [2, 3, 20, 29]. Em particular, a regra $\{x\} \Rightarrow \{y\}$ aponta para um acoplamento de mudança de y para x, indicando que a evolução de y está acoplada com a evolução de x.

Acoplamentos de mudança se tornaram um componente fundamental de diversas técnicas e ferramentas de recomendação (ou predição) de mudanças [5, 17]. Entretanto, existem questões em aberto em relação a aspectos práticos da aplicação desses acoplamentos para a recomendação de mudanças conjuntas [20]. Em particular, o tamanho do histórico do projeto, assim como a escolha dos limiares para suporte e confiança, interferem na qualidade das regras geradas [19, 29].

Utilizar um histórico muito curto pode resultar em regras que não expressam conhecimento suficiente sobre o sistema. No entanto, um histórico muito longo pode conter informações desatualizadas e inserir ruídos nas regras geradas. Segundo Zimmermman et al. [29], há um custo-benefício entre a quantidade de recomendações e a qualidade dessas recomendações. Usar valores baixos de suporte e

confiança possibilita uma maior quantidade de regras de associação, mas com menor acurácia de recomendações.

Zimmermman et al. [29] sugerem um limiar de suporte igual a 3 e confiança de 90%. Valores similares podem ser encontrados nos trabalhos de DiPenta et al. [3] e Bavota et al. [2], que utilizam limiar de confiança de 80% e de suporte entre 2 e 3. Os trabalhos de Moonen et al. [18, 19], embora tenham avaliado empiricamente as regras de associação, não definem limiares ótimos para a aplicação em diferentes projetos.

Determinar limiares ótimos das medidas de interesse e o tamanho do histórico ainda é uma tarefa difícil. Deste modo, é necessário que novas técnicas determinem a configuração ideal para que os desenvolvedores possam se valer de acoplamentos evolucionários na prática.

4 TRABALHOS RELACIONADOS

A literatura de mineração de repositório de software frequentemente menciona que a escolha do tamanho do histórico usado interfere na acurácia dos resultados [9, 11, 29]. Isso ocorre tanto ao escolher um histórico extremamente pequeno ou extremamente grande, seja porque não há informações suficientes para gerar conhecimento sobre o sistema, ou porque algumas informações já estão desatualizadas. Moonen et al. [19] investigou o impacto desses efeitos e gerou uma função de regressão linear, que encontra o tamanho de histórico ideal para ser usado como treinamento. Entretanto, foram considerados apenas projetos com mais de $5k$ transações no histórico.

Pesquisadores de mineração de regras de associação [16, 28] expõem a necessidade de se investigar a influência dos parâmetros dos algoritmos. Considerando a dificuldade que usuários possuem em especificar um limiar apropriado para as regras de associação, Selvi et al. [25] propõem uma modificação para o algoritmo *Apriori* de forma que limiares de suporte são definidos automaticamente para cada nível do processo de geração do conjunto de itens frequentes ("frequent itemset").

Moonen et al. [18] avaliaram, além de suporte e confiança, outras 38 medidas de interesse para recomendação de software baseada em acoplamento evolutivo. Segundo os autores, suporte e confiança estão entre as melhores medidas de interesse para definir bons acoplamentos de mudança. Os resultados de Moonen et al. também mostram que as recomendações com melhores precisões são obtidas usando transações relativamente pequenas, contendo entre quatro e seis arquivos.

No entanto, determinar os limiares ideais para que as regras sejam suficientemente relevantes é uma tarefa complicada e depende das características do projeto em questão. No trabalho de Zimmermann et al. [29], por exemplo, apenas regras de associação com suporte maior que 1 e confiança maior que 0.5 são consideradas relevantes. Por sua vez, Bavota et al. [2] consideraram relevante regras com limiar de confiança 0.8 e suporte 0.2. No entanto, nenhum trabalho anterior relata como escolher automaticamente um limiar de suporte e confiança, e também um conjunto de treinamento para um projeto de software a fim de otimizar as recomendações geradas.

Pesquisadores de Engenharia de Software têm utilizado algoritmos genéticos em variados cenários. Por exemplo, na área de testes, algoritmos genéticos foram utilizados para gerar e avaliar um conjunto de teste para linha de produto de software a partir de parâmetros como operadores de mutação [6]. Colares et al. [4] usaram um algoritmo genético multiobjetivo para mostrar a aplicabilidade de sua abordagem para o problema de planejamento de *realease* de software.

5 METODOLOGIA

Nesta seção são apresentadas as questões de pesquisa, a abordagem proposta, os cenários de avaliação e os projetos utilizados para comparar a abordagem proposta com o trabalho de Moonen et al. [19].

5.1 Questões de Pesquisa

Este trabalho propõe o uso do Algoritmo Genético para otimizar a seleção do melhor conjunto de treinamento e dos limiares de suporte e confiança para um projeto de software, conforme problematizado na Seção 3. Para tal, investigamos empiricamente duas questões de pesquisa:

(QP1) Como a acurácia do modelo de recomendação de mudanças baseado em Algoritmo Genético se compara àquela do modelo estático proposto por Moonen et al.?

A primeira questão de pesquisa visa comparar a acurácia do modelo de recomendação de mudanças baseado em Algoritmo Genético com o modelo estático proposto por Moonen et al. [19]. Este modelo utiliza uma função de regressão que define o tamanho ideal do conjunto de treinamento baseado no número de arquivos e na média do tamanho dos *commits* do projeto. Como Moonen et al. [19] não indica quais foram os limiares de suporte e confiança usados, nós comparamos o modelo genético proposto neste trabalho com o modelo de regressão de Moonen et al. [19] usando limiares de suporte e confiança sugeridos na literatura [2, 29].

O objetivo desta questão de pesquisa é verificar se o modelo de recomendação proposto possui maior acurácia do que o modelo proposto por Moonen et al. [19]. Nesta QP foram testados a acurácia dos modelos para prever as 5% mais recentes modificações realizadas em cada um dos cinco projetos.

(QP2) Como a acurácia dos modelos de recomendação de mudança se comporta quando o conjunto de teste aumenta?

A mudança de um software é um processo inevitável segundo a primeira lei de Lehman [15]. O ambiente muda constantemente, surgem novos requisitos e o software deve ser modificado para não se tornar progressivamente menos satisfatório. Nesta questão de pesquisa, investigamos o quanto novas mudanças no sistema deterioram a estabilidade do modelo de recomendações de mudanças. Para isso, testamos incrementalmente novas mudanças no período de teste e analisamos o impacto na acurácia das recomendações geradas.

Para esta análise foram usados quatro períodos de teste, sendo eles 5%, 10%, 20% e 30% mais recente das transações do histórico de mudanças do projeto. O objetivo desta questão de pesquisa é verificar se a variação do tamanho do conjunto de teste de 5% até 30% afeta a estabilidade dos modelos fazendo com que eles percam acurácia. Se isso ocorrer, podemos concluir que usar o modelo proposto neste trabalho é melhor, uma vez que ele conseguiria otimizar

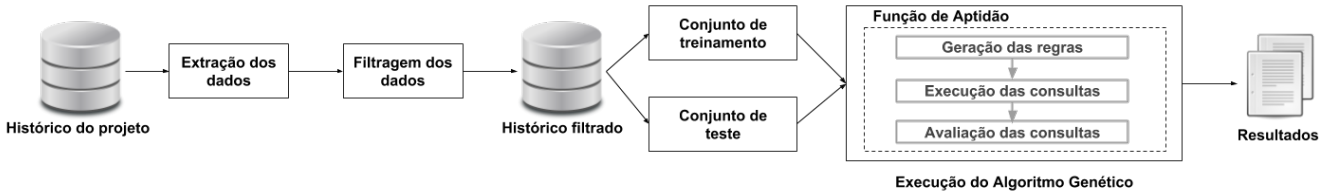

Figura 1: Visão geral da abordagem de recomendação de mudanças baseada em Algoritmo Genético.

as recomendações em diferentes cenários sem a necessidade de realizar um estudo empírico para definir qual deveria ser o limiar de suporte e confiança que seria usado combinado com o tamanho do histórico de modificações sugerido pela regressão proposta por Moonen et al. [19].

5.2 Projetos Estudados

Para avaliar a abordagem proposta em várias condições, foram selecionados cinco projetos de código aberto com diferentes tamanhos de histórico de transações. Além de possuírem históricos de tamanhos distintos, esses projetos apresentam frequência de *commits*, linguagem e domínio diferentes. A Tabela 1 apresenta as características relevantes dos projetos utilizados na avaliação e sua diversidade.

Entre os projetos selecionados, estão três *framework web*, Laravel, Django e Shiny, e também a biblioteca Gson criada pela Google. O quinto projeto escolhido é o Cpython, a principal implementação da linguagem de programação Python, escrita em C. Os cinco projetos têm seu código fonte disponível no GitHub.

A Tabela 1 mostra que o tempo de histórico dos projetos selecionados varia de pequeno a grande [18, 19]. Trabalhos anteriores consideram apenas sistemas com histórico que variam de médio a grande. Gson é o menor projeto, com aproximadamente nove anos de histórico, 516 arquivos únicos e 2533 *commits* até a coleta dos dados.

5.3 Abordagem Proposta

Nesta seção, apresentamos a abordagem de recomendação de mudanças baseada em Algoritmo Genético. Tal abordagem visa automatizar as recomendações de limiares de suporte e confiança para um projeto de software. A Figura 1 mostra uma visão geral da abordagem.

Definição da Abordagem. Para construir a abordagem proposta, utilizamos um Algoritmo Genético. Algoritmo Genético (AG) [14] é uma técnica de busca e otimização inspirada nos princípios da evolução e seleção natural. Os AGs fornecem um mecanismo de busca que pode ser usado tanto em problemas de classificação, quanto em problemas de otimização. O AG simula o processo de evolução natural, ou biológica, no qual os indivíduos mais aptos dominam sobre os mais fracos imitando mecanismos biológicos de evolução, tais como seleção natural, cruzamento e mutação.

Uma busca com AG começa com uma população aleatória de soluções, na qual cada indivíduo representa uma solução para o problema. A população evolui para melhores soluções por meio de gerações subsequentes e, durante cada geração, os indivíduos são avaliados com base em sua função de aptidão, de modo que apenas os indivíduos mais aptos se reproduzem. Para criar a próxima geração, informações genéticas são transferidas para os descendentes. Os novos indivíduos são gerados aplicando um operador de seleção com base na aptidão dos indivíduos a serem reproduzidos, recombinando com uma dada probabilidade, dois indivíduos da geração atual por meio de cruzamento e modificando, com uma determinada probabilidade, indivíduos por meio de mutação. O processo de evolução é encerrado com base em critérios de convergência, geralmente um número máximo de gerações. Alternativamente, o processo de evolução é interrompido quando um grande número de gerações não apresenta melhoria no melhor valor de aptidão, ou quando um valor predefinido é atingido.

Em um AG, a função de aptidão confere uma nota para cada indivíduo de acordo com a sua aptidão. Assim, o objetivo é maximizar o valor da função de aptidão de modo que a cada nova geração, os indivíduos estejam mais próximos do ótimo global, que é a melhor solução para o problema. Utilizamos AG para explorar efetivamente o espaço de busca de possíveis combinações de tempo de treinamento, suporte e confiança para selecionar as melhores recomendações de mudanças esperadas para um projeto de software.

5.3.1 ***Pré-processamento.*** O pré-processamento dos dados consiste nas seguintes etapas.

Extração dos dados. Nesta etapa, clonamos o repositório de código fonte e recuperamos as informações de todo o histórico de *commits*, como os arquivos que foram modificados, o autor e data de cada *commit*. Essa etapa resulta em um arquivo CSV com os dados coletados do histórico de *commits* do projeto. Para desempenhar tal tarefa utilizamos o RepoDriller[1], um framework Java que auxilia na mineração de repositórios de software e possibilita a extração de informações de um repositório Git e a exportação para arquivos CSV.

Filtragem dos dados. Removemos do histórico do projeto *commits* com mais de trinta arquivos, assim como no trabalho de Zimmermann et al. [29], e também *commits* apenas com imagens. Essas filtragens removem grandes transações que não apresentam relevância para se obter os acoplamentos de mudança dos arquivos [18] e mudanças que não envolvem código-fonte. A filtragem dos *commit* resulta em um conjunto de transações $T = \{t_1, t_2, \cdots, t_n\}$, sendo n o número de *commits* que permaneceram no histórico após a filtragem.

[1] https://github.com/mauricioaniche/repodriller

Tabela 1: Características dos projetos estudados

Projeto	URL do GitHub	Linguagem usada	Tamanho médio das transações	Arquivos únicos	Histórico de versão minerado	Tamanho do histórico (em meses)	Número de transações	Média de transações por mês
Gson	https://github.com/google/gson	Java	3.13	516	24-04-2008 – 16-07-2017	111	2533	22.81
Shiny	https://github.com/rstudio/shiny	R	2.68	567	20-06-2012 – 13-07-2017	61	3789	62.11
Laravel	https://github.com/django/django	PHP	1.91	492	09-06-2011 – 16-08-2016	62	4943	79.72
Django	https://github.com/laravel/laravel	Python	2.52	2200	12-07-2005 – 16-08-2016	133	33532	252.12
CPython	https://github.com/python/cpython	C	2.07	6703	09-08-1990 – 07-05-2016	309	87171	282.10

Separação do conjunto de treinamento e teste. Dado o conjunto de transações T, dividimos T em um conjunto de treinamento $Treinamento = \{t_{n_1}, t_{n_2}, \cdots, t_{n_s}\}$ e um conjunto de teste $Teste = \{t_{s+1}, \cdots, t_n\}$. O conjunto de teste é extraído das transações mais recentes de T e então geramos o conjunto de treinamento com a porcentagem restante. Aleatorizamos a ordem dos arquivos em cada uma das transações dos conjuntos de treinamento gerados, pois não sabemos qual o primeiro arquivo modificado pelo desenvolvedor.

5.3.2 *Execução do Algoritmo Genético.*

Função de Aptidão. Modelamos uma função de aptidão para o AG para otimizar recomendações de mudanças. Essa função de aptidão utiliza uma porcentagem do conjunto treinamento para construir o modelo, gerando regras de associação, e utiliza também as transações do conjunto de teste para para formar consultas e avaliar as recomendações geradas pelas regras, similar à avaliação feita por outros pesquisadores [18, 19, 29]. A otimização realizada pelo AG é baseada no valor de Precisões Médias (AP), uma métrica que avalia o desempenho de uma recomendação.

A definição da função de aptidão é um passo importante da construção do AG. Para o problema que estamos otimizando, as potenciais soluções são representadas por uma porcentagem do conjunto de treinamento, um limiar de suporte e um limiar de confiança chamados respectivamente de *porcentagem_treinamento*, *suporte* e *confianca*. Além destes três parâmetros, o conjunto $Teste$ e o conjunto $Treinamento$ são passados como parâmetros adicionais para a função de aptidão e são mantidos inalterados durante a busca. Como mostrado no Algoritmo 1 e na Figura 1, a função de aptidão abrange três passos: geração das regras, execução das consultas e avaliação das consultas. Cada um desses passos são descritos a seguir.

Geração das regras. O primeiro passo da função de aptidão é extrair do conjunto $Treinamento$ a porcentagem de transações correspondente à potencial solução *porcentagem_treinamento*. Um novo conjunto de treinamento $Treinamento_n$ é formado com as transações extraídas. A partir de $Treinamento_n$, as regras de associação são geradas com as medidas de interesse *suporte* e *confiança*.

Para minerar as regras de associação, utilizamos o algoritmo Apriori [1]. Esse algoritmo requer um valor de suporte e confiança mínimo como entrada, de modo que itens que não são frequentes são removidos preventivamente. Primeiro, o Apriori encontra todos os conjuntos de itens (*itemsets*) com suporte maior que o suporte mínimo. A partir de cada um desses itemsets obtidos são criadas regras. Todas as regras criadas a partir de um único itemset têm o mesmo suporte, porém somente as regras que estão acima da confiança mínima são retornadas.

Algorithm 1: FuncaoAptidao

Input: porcentagem_treinamento, suporte, confianca
Output: O valor de MAP correspondente as recomendações

1 {Geração das regras}
2 $treinamento\_n \leftarrow extrai\_transacoes(treinamento,\ porcentagem\_treinamento)$
3 $R \leftarrow arules(treinamento\_n,\ suporte,\ confianca)$
4 **for** $ft \in Teste$ **do**
5 {Execução da consulta}
6 $Q \leftarrow ft[1]$
7 $E \leftarrow ft - Q$
8 $R_p \leftarrow aplica\_consulta(R,\ Q)$
9 $R_p \leftarrow ordena(R_p,\ 10)$ {ordenação usando suporte e k=10}
10 $F \leftarrow consequente(R_p)$
11 {Avaliação da consulta}
12 $ap \leftarrow calcula\_ap(E,\ F)$
13 **end**
14 $map \leftarrow media(ap)$
15 **return** map

Seguindo Zimmermann [29], computamos um conjunto de regras de associação R com um único item em seu consequente $X \Rightarrow \{e\}$. Tais regras são suficientes, porque nessa abordagem consideramos a união dos consequentes das regras com antecedente X.

Execução das consultas. Após a geração das regras, a função de aptidão verifica para cada uma das transações de $Teste$ se seus itens são previstos a partir de $Treinamento_n$. Para desempenhar essa tarefa, geramos uma consulta $q = (Q, E)$ para cada uma das transações F_t. Sendo $F_t = \{f_1, \cdots, f_n\}$ uma transação pertencente ao conjunto $Teste$, a particionamos em uma consulta $Q = f_1$ e um resultado esperado $E = F_t - Q$.

Ao aplicarmos uma consulta ao conjunto de regras de associação R, obtemos o conjunto de regras R_p referente a Q. Isso significa que R_p contém apenas regras nas quais a consulta é o antecessor: $Q \Rightarrow \{e\}$. As regras R_p, no cenário prático, representam uma lista de sugestão que o desenvolvedor recebe após selecionar o arquivo Q. Assumindo que listas muito grandes não são viáveis no cenário prático por apresentarem ao desenvolvedor muitas sugestões e o forçarem a navegar por muitos arquivos, consideramos apenas as primeiras 10 regras. Selecionamos a partir de R_p as $top - k$ regras, ordenadas por valor de confiança. Após essa filtragem por valor de confiança, R_p apresenta tamanho menor ou igual a 10.

Avaliação das consultas. Aplicar a consulta $q = (Q, E)$ resulta em uma lista de recomendação ordenada F, a qual corresponde aos

consequentes das regras R_p. Ao comparar essa lista de arquivos com os resultados esperados, obtemos o conjunto de recomendações corretas, estão em F e também correspondem ao resultado esperado, e também o conjunto de recomendações incorretas, estão entre os arquivos em F mas não correspondem ao resultado esperado.

A capacidade das regras de associação de produzir recomendações precisas e completas pode ser medida por meio de *Precisão* e *Sensibilidade* [27]. A *Precisão* de uma recomendação é a razão entre o número de itens corretos e o total de itens recomendados. A *Sensibilidade*, por sua vez, é a razão entre o número de itens corretos e o total de itens esperados.

A última etapa da função de aptidão é devolver o valor de aptidão do indivíduo. Trabalhos anteriores [3, 29] utilizaram *Precisão* e *Sensibilidade* como métricas de desempenho. Entretanto, a avaliação da execução de uma consulta foi realizada por meio de Precisão Média (AP) [27].

De acordo com o trabalho de Rolfsnes et al. [23], precisão média é a métrica mais indicada para avaliar listas de recomendações. Essa métrica leva em consideração a ordem dos arquivos recomendados na lista de resultados esperados. Assim, uma recomendação correta no início da lista é mais provável de ser considerada pelo desenvolvedor. Devido ao contexto, a métrica precisão média foi escolhida para que a posição ocupada pelo arquivo recomendado seja levada em consideração na avaliação da recomendação.

Dado um resultado esperado E para uma consulta Q e lista de recomendação ordenada F, precisão média (AP) é definida como:

$$AP(F) = \sum_{k=1}^{F} P(k) * \Delta r(k)$$

Onde, $P(k)$ é a precisão calculada nos primeiros k itens da lista e $\Delta r(k)$ é a diferença entre a Sensibilidade calculada nos $k^{\underline{o}} - 1$ arquivos e a Sensibilidade calculada nos $k^{\underline{o}}$ arquivos.

Como medida de desempenho global, usamos Média das Precisões Médias (MAP) sobre o conjunto de todas as consultas executadas. O valor de MAP representa a aptidão do indivíduo.

5.4 Cenário de Avaliação

Os resultados reportados na Seção 6 e a função de aptidão apresentada em 5.3 foram implementadas utilizando o ambiente estatístico R. Para execução do AG foi escolhido o pacote GA [24] e para gerar as regras de associação, o pacote arules [10]. A nossa implementação da função de aptidão está disponível online[2].

Configurando o Algoritmo Genético. Quanto às configurações do AG, usamos uma probabilidade de cruzamento de 0.8 e uma probabilidade de mutação de 0.1. Os operadores de *crossover* e mutação utilizados foram os operadores *local arithmetic crossover* e *uniform random mutation* implementados pelo pacote GA.

Determinamos quatro tamanhos diferentes para a população: 25, 50, 100 e 200 indivíduos. O objetivo foi avaliar o impacto do tamanho da população na otimização das recomendações de mudanças, a fim de definir a quantidade de indivíduos necessária para oferecer total cobertura do espaço de busca. A população ideal para um determinado problema é quando há um equilíbrio entre um baixo número de gerações para a convergência e uma maior precisão à medida que a população aumenta [8].

Após executar o AG para os cinco projetos utilizando 5% do histórico para teste e os quatro tamanhos de população preestabelecidos, escolhemos uma população de 200 indivíduos para utilizar nas demais execuções. Essa escolha deve-se ao fato de termos observado uma convergência mais rápida para o ótimo global ao utilizar uma população de 200 indivíduos no problema em questão. Como condição de parada para o AG, terminamos a evolução após atingir o número máximo de 100 gerações.

A Figura 2 apresenta a evolução da população do AG para o projeto Gson. O algoritmo proposto evoluiu e convergiu rapidamente para a melhor solução, o que indica que a configuração da população do algoritmo está correta e por falta de espaço não são apresentados os gráficos para os outros quatro projetos, contudo, o mesmo comportamento foi observado [3].

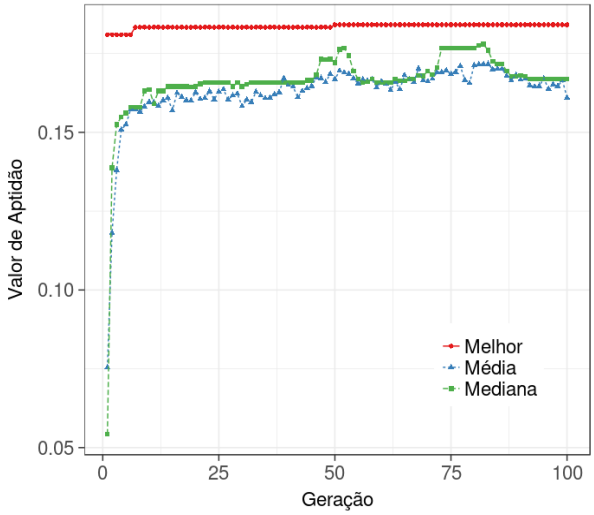

Figura 2: Execução do AG.

Restrição do espaço de busca. Para limitar o espaço de busca do AG e definir o intervalo de variação, informamos um limite máximo e mínimo para cada uma das variáveis de decisão. Para a variável de decisão que representa a porcentagem de treinamento, adotamos o limite mínimo como 10% e variamos o limite máximo em 95%, 90%, 80% e 70%. Com isso, variamos o teste em 5%, 10%, 20% e 30% do histórico. Para a variável de decisão confiança, que corresponde à confiança mínima das regras geradas, estabelecemos 0.1 como limite mínimo e 1 para o limite máximo. Esses valores são empregados nos trabalhos que calculam acoplamento conceitual [3, 23, 29].

Os limites da variável de decisão suporte, que corresponde ao suporte mínimo das regras geradas, são determinados de acordo com a quantidade de transações disponíveis no conjunto de treinamento de cada projeto. O suporte mínimo é obtido por $1 \div Tamanho\ do\ treinamento$. O suporte máximo é obtido por $20 \div Tamanho\ do\ treinamento$.

[2]https://github.com/mairieli/fitness-recommendation

[3]Os gráficos podem ser encontrados em: https://github.com/mairieli/fitness-recommendation/tree/master/plot

Tabela 2: Valores de Suporte mínimo e Suporte máximo

Projeto	Suporte mínimo	Suporte máximo
CPython	0.00001	0.0002
Django	0.00003	0.0006
Laravel	0.0002	0.004
Shiny	0.0002	0.005
Gson	0.0004	0.008

A Tabela 2 mostra os valores de suporte mínimo e máximo adotados para os projetos selecionados. O AG utiliza os limites mínimos e máximos para limitar seu espaço de busca ao gerar novos indivíduos.

Avaliação da acurácia da abordagem baseada em algoritmo genético. Para avaliar o desempenho do AG em encontrar um conjunto de treinamento, suporte e confiança capazes de otimizar recomendações de mudanças para um projeto de software, utilizamos o trabalho de Moonen et al. [19]. Os autores definem uma função de regressão baseada no número de arquivos e média do tamanho dos *commits* do projetos para prever o valor do tamanho do histórico que deve ser usado como conjunto de treinamento. A função de regressão é dada por:

$$\begin{aligned} Tamanho\ do\ treinamento = {} & 33974.1 \\ & + 208.4 \times Número\ de\ arquivos\ (em\ 1000) \\ & - 3958.9 \times Média\ do\ tamanho\ dos\ commits \end{aligned} \tag{1}$$

Contudo, o trabalho de Moonen não apresenta quais são os limiares de suporte e confiança que devem ser usados. Por este motivo, comparamos nossa abordagem com um modelo estático gerado a partir do tamanho do histórico vindo do trabalho de Moonen et al. [18], variando os limiares de suporte entre 2 a 20 e os valores de confiança em 0.10, 0.5 e 0.9, de acordo com as recomendações de Zimmermann et al. [29] e Bavota et al. [2].

A ideia-chave por detrás da avaliação proposta para responder **QP1** consistiu em executar os dois modelos, utilizando 5% do histórico mais recente para teste, e selecionar os melhores resultados para análise. Desta forma, o algoritmo genético utiliza os 95% restante das transações para selecionar o treinamento necessário para otimizar as recomendações, enquanto a função de regressão define o tamanho do treinamento independente do tamanho do teste usado.

Avaliação da estabilidade dos modelos. Para investigar a **QP2**, avaliamos o quanto novas mudanças no sistema deterioram a estabilidade do modelo de recomendações de mudanças. Foram executados os dois modelos de recomendação de mudanças, usando quatro tamanhos diferentes para o conjunto de teste: 5%, 10%, 20% e 30% das transações mais recentes do histórico de mudanças do projeto. Cada uma das execuções dos modelos considerava apenas um destes valores fixos de teste. Assim, para um valor fixo de teste o AG pode escolher entre a porcentagem restante de transações no histórico de mudanças (conjunto de treinamento) o tamanho do conjunto usado para gerar as recomendações. Da mesma forma, a regressão definiu a quantidade de transações do conjunto de treinamento que seriam usadas. Após a execução dos dois modelos, os melhores resultados para cada um dos períodos de teste foram selecionados para análise.

6 RESULTADOS

Nesta seção, apresentamos os resultados da execução do AG para otimizar a recomendação de mudanças com o comparativo realizado com a função de regressão proposta por Moonen et al. [19]. Também apresentamos a investigação da avaliação da deterioração dos modelos à medida que o conjunto de treinamento diminui.

(QP1) Como a acurácia do modelo de recomendação de mudanças baseado em algoritmo genético se compara àquela do modelo estático proposto por Moonen et al.?

Para responder esta questão de pesquisa foram executados os dois modelos para testar os 5% de modificações mais recentes de cada um dos projetos analisados. O AG utilizou os 95% restante das transações para selecionar o tamanho do histórico necessário para otimizar as recomendações de mudança.

Durante a execução do AG, observamos que os conjuntos de treinamento selecionados correspondem a um total de 35%, 41%, 13%, 13% e 79% das transações dos projetos CPython, Django, Laravel, Shiny e Gson respectivamente, como pode ser visto na Tabela 3.

A função de regressão de Moonen et al. [19] não é capaz de prever a quantidade de histórico necessário para gerar ótimas recomendações de mudança em projetos pequenos. Nos projetos com menos de cinco mil transações, a quantidade de transações de treinamento recomendada pela regressão foi maior do que o tamanho do histórico disponível. A Tabela 3 compara os resultados de tamanho de treinamento obtidos. Pode-se observar que o AG possui a vantagem de se adaptar ao tamanho do projeto e ao mesmo tempo selecionar um conjunto de treinamento capaz de otimizar as recomendações de mudanças geradas a partir dele.

A Figura 3 apresenta a variação dos valores de MAP obtidos com as diferentes configurações de suporte e confiança. É possível perceber que, há uma variabilidade na acurácia das recomendações independente da quantidade de *commits* do projeto. É importante mencionar que a configuração manual dos valores de suporte e confiança podem levar a até 30% de diferença na acurácia dos modelos, como pode ser observado na variabilidade do valor de MAP, tendo em vista que todas as execuções para um projeto usaram o mesmo tamanho de treinamento indicado pela função de regressão. Essa grande diferença de resultados reforça a necessidade de experimentação de diferentes parâmetros, o que é tratado de modo automático pela abordagem proposta.

Comparando a acurácia do modelo proposto neste trabalho com o trabalho de Moonen, observamos que a diferença percentual da precisão de acerto nas recomendações entre as duas abordagens é muito pequena. No Projeto CPython, a abordagem proposta selecionou 29k *commits* para o treino enquanto a abordagem do Moonen selecionou 27k. O limiar de suporte escolhido pelas duas abordagens foi o mesmo, ocorrendo uma pequena variação de 0.01 na confiança. Neste caso, ter adicionado mais histórico para o treinamento não proporcionou melhora no valor de MAP quando comparamos as duas abordagens.

Para o projeto Django, a abordagem proposta selecionou $11k$ transações a menos. Embora o limiar de suporte e confiança tenham sido os mesmos, o AG apresentou uma pequena melhora (0.013%)

Tabela 3: Melhores resultados para 5% de teste.

Projeto	Modelo	Teste (%)	Transações no treinamento	Transações usadas no treinamento	Transações usadas no teste	Suporte	Confiança	MAP
CPython	Regressão	5	82813	27176	2083	2	0.10	0.298
CPython	AG	5	82813	29622	2083	2	0.12	0.299
Django	Regressão	5	31856	24456	845	2	0.10	0.183
Django	AG	5	31856	13317	845	2	0.10	0.196
Laravel	Regressão	5	4696	26515 *	54	2	0.10	0.134
Laravel	AG	5	4696	637	54	10	0.12	0.157
Shiny	Regresão	5	3600	23482*	116	2	0.15	0.267
Shiny	AG	5	3600	498	116	5	0.12	0.342
Gson	Regresão	5	2407	21690*	59	2	0.10	0.192
Gson	AG	5	2407	1903	59	1	0.17	0.196

* O tamanho do treinamento previsto pela função de regressão é maior que o tamanho do histórico do projeto, então foram usadas as transações disponíveis.

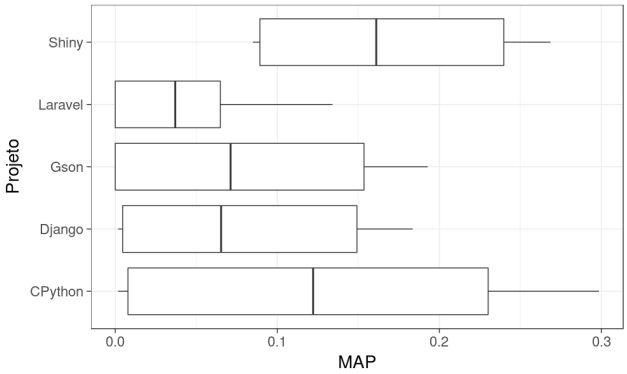

Figura 3: Distribuição do modelo estático para cada projeto.

na acurácia das recomendações. Para os projetos Laravel, Shiny e Gson, a abordagem de Moonen et al. indicou que deveriam ser selecionadas respectivamente $27k$, $24k$ e $22k$ transações, todavia essas quantidades são maiores do que o histórico disponível desses projetos. Neste caso, selecionamos a quantidade total restante do conjunto de treinamento para realizar a comparação. Para o projeto Shiny o AG selecionou somente 498 transações e neste cenário verificamos a maior diferença na acurácia entre os modelos (0.075%).

O AG proposto otimizou automaticamente o limiar de suporte e confiança e o tamanho de treinamento para projetos de diferentes tamanhos. A função de regressão proposta por Moonen et al. não foi capaz de prever a quantidade de histórico para os projetos menores (Laravel, Shiny e Gson). Apesar da pouca diferença no valor de MAP, a melhoria pode ser relevante, uma vez que um maior valor de MAP diminui o esforço do desenvolvedor em encontrar os arquivos para realizar uma mudança.

(QP2) Como a acurácia dos modelos de recomendação de mudança se comporta quando o conjunto de teste aumenta?

Inserir novas mudanças no software deteriora a estabilidade do modelo de recomendações de mudanças. Nesta questão de pesquisa comparamos os modelos em quatro configurações de teste distintas. A Tabela 3 apresenta os resultados para 5% de teste, e a Tabela 4 apresenta os resultados obtidos com 10%, 20% e 30% de teste para cada modelo.

O maior projeto estudado, CPython, apresenta os melhores resultados de recomendação de mudanças ao usar, nos dois modelos, 10% de seu histórico para teste. Aumentar o tamanho do teste de 10% para 20% e 30% fez a acurácia das recomendações geradas diminuir entre 0.03 a 0.05. Para o CPython, mesmo aumentando o conjunto de teste, os modelos praticamente não deterioraram.

O projeto Django apresentou uma pequena diferença em percentual na acurácia das recomendações para as execuções com os diferentes tamanhos de teste. A diferença, para os dois modelos foi de aproximadamente 0.01. Django é considerado neste trabalho como um projeto de tamanho médio, possuindo $34k$ transações em seu histórico. Enquanto a abordagem de Moonen et al. selecionou $24k$ transações de treinamento, valor esse insuficiente para a quantidade de transações disponíveis ao usar 30% de teste. Nossa abordagem selecionou cerca de $10k$ transações a menos.

Para o projeto Laravel, aumentar o tamanho do teste de 10% para 20% fez com que a estabilidade das recomendações tanto no modelo baseado em AG, quanto no estático se deteriorasse, diminuindo o valor de MAP em 0.077 nos dois modelos. O mesmo pode ser observado para os projetos menores Shiny e Gson. Aumentar o tamanho do teste de 10% para 20% para o Shiny impactou na diminuição de cerca de 0.089 no valor de MAP na abordagem proposta e 0.109 ao usar o modelo estático. Isto se deve ao fato de que projetos pequenos possuem pouco histórico de transações disponível, e aumentar o conjunto de teste significa limitar consideravelmente o tamanho do conjunto de treinamento disponível.

Laravel possui apenas 492 arquivos únicos. Limitar o conjunto de treinamento possivelmente reduz a quantidade de mudanças conjuntas entre os arquivos, bem como a quantidade de arquivos que mudaram pelo menos uma vez no período usado para criar o conjunto de treinamento.

Apesar dos resultados apresentarem pouca diferença na acurácia (valor de MAP) das recomendações, a redução no valor de MAP impacta o esforço que o desenvolvedor tem para encontrar os arquivos na lista de recomendações, uma vez que o MAP está diretamente relacionado com a posição que os arquivos corretos ocupam na lista de recomendação. Dessa forma, se o MAP diminuir muito, o

Tabela 4: Melhores resultados para diferentes porcentagens de teste.

Projeto	Modelo	Teste (%)	Transações no treinamento	Transações usadas no treinamento	Transações usadas no teste	Suporte	Confiança	MAP
CPython	Regressão	10	78454	27176	4188	2	0.10	0.306
CPython	AG	10	78454	38627	4188	1	0.10	0.316
CPython	Regressão	20	69737	27176	8286	2	0.10	0.271
CPython	AG	20	69737	33081	8286	1	0.10	0.276
CPython	Regressão	30	61019	27176	11977	2	0.10	0.258
CPython	AG	30	61019	25222	11977	2	0.11	0.258
Django	Regressão	10	30179	24456	1703	2	0.10	0.175
Django	AG	10	30179	14351	1703	1	0.10	0.187
Django	Regressão	20	26826	24456	3389	2	0.10	0.171
Django	AG	20	26826	9926	3389	2	0.10	0.180
Django	Regressão	30	23473	24456 *	5059	2	0.10	0.176
Django	AG	30	23473	7613	5059	2	0.10	0.187
Laravel	Regressão	10	4449	26515 *	102	2	0.10	0.111
Laravel	AG	10	4449	659	102	7	0.13	0.120
Laravel	Regressão	20	3955	26515 *	253	2	0.10	0.034
Laravel	AG	20	3955	1296	253	5	0.10	0.043
Laravel	Regressão	30	3461	26515 *	353	2	0.10	0.035
Laravel	AG	30	3461	674	353	3	0.12	0.047
Shiny	Regresão	10	3411	23482*	227	2	0.10	0.322
Shiny	AG	10	3411	3033	227	2	0.12	0.319
Shiny	Regresão	20	3032	23482*	464	2	0.10	0.213
Shiny	AG	20	3032	1054	464	2	0.11	0.230
Shiny	Regresão	30	2633	23482*	683	2	0.10	0.209
Shiny	AG	30	2633	872	683	2	0.10	0.225
Gson	Regresão	10	2280	21690*	114	2	0.10	0.184
Gson	AG	10	2280	1771	114	2	0.10	0.181
Gson	Regresão	20	2027	21690*	219	2	0.10	0.145
Gson	AG	20	2027	1231	219	2	0.12	0.155
Gson	Regresão	30	1774	21690*	326	2	0.10	0.180
Gson	AG	30	1774	979	326	3	0.12	0.191

* O tamanho do treinamento previsto pela função de regressão é maior que o tamanho do histórico do projeto, então foram usadas as transações disponíveis.

esforço do desenvolvedor será muito maior, porque ele receberá muitos falsos positivos.

Escolher valores arbitrários para as configurações das medidas de interesse que otimizam as recomendações geradas requer muito esforço. Como inserir novas mudanças no sistema deteriora a estabilidade do modelo de recomendações, principalmente em projetos pequenos, o esforço de encontrar essas medidas de interesse se torna constante se não houver uma técnica que automatize essa escolha. Por este motivo, usar um AG para automatizar a escolha dos valores das medidas de interesse que otimizam as recomendações é importante.

Inserir novas mudanças no sistema deteriora a estabilidade do modelo de recomendações de mudanças. Portanto, usar o modelo proposto é melhor em relação ao modelo estático, uma vez que os limiares de suporte e confiança tendem a mudar com o tamanho de treinamento disponível.

7 AMEAÇAS À VALIDADE

Hábitos de Commit. A abordagem apresentada na Seção 5.3 está baseada na mineração de regras de associação que utiliza o histórico de modificações registradas no controle de versão. Entretanto, desenvolvedores podem realizar *commits* que envolvam arquivos não relacionados e que portanto não deveriam ser modificados conjuntamente [20]. Esse conceito é conhecido na literatura como *tangled changes* [13] e pode influenciar a geração das regras de associação usadas pelos modelos de recomendações de mudança. Para evitar esse viés, removemos *commits* que continham mais de 30 arquivos [29], já que eles poderiam se referir a operações de mudança de ramos do controle de versão ou múltiplas mudanças.

Amostragem aleatória. O experimento realizou consultas a partir de um arquivo escolhido aleatoriamente. Embora tenhamos usado a mesma amostra para comparar os dois modelos, há a possibilidade de que se a consulta fosse realizada a partir de um outro arquivo selecionado aleatoriamente poderia apresentar outro valor de MAP. Essa escolha é feita porque não se sabe qual foi o primeiro arquivo modificado pelo desenvolvedor durante a realização do *commit*. Essa abordagem também foi utilizada na avaliação da função de regressão proposta por Moonen et al. [19].

Generalização. Realizamos os experimentos em cinco projetos de código aberto. Os cinco projetos selecionados variam em tempo, tamanho de histórico e frequência de transações. Apesar de sabermos que é necessário aumentar a quantidade de projetos para generalizar a comparação entre as abordagens, o uso de um projeto com pequena quantidade de transações no histórico de mudanças adicionou uma perspectiva diferente de avaliação dos resultados

que não foi avaliada por Moonen et al. [19]. Também nos possibilitou realizar inspeções e ter maior controle sobre o correto funcionamento da abordagem.

Implementação do Algoritmo Genético. As configurações do algoritmo genético podem influenciar os resultados obtidos. Entretanto, verificou-se que todo o espaço de busca foi explorado, bem como a melhor escolha da população foi obtida, uma vez que o algoritmo genético convergiu rapidamente para a melhor solução de escolha do conjunto de treinamento e limiares de suporte e confiança.

8 CONCLUSÕES

Neste trabalho, investigamos empiricamente como determinar valores ótimos de suporte, confiança e conjunto de treinamento que geram as melhores recomendações de mudança. Para isto, definimos uma função de aptidão para um AG que seleciona o melhor conjunto de treinamento e valores de suporte e confiança que otimizam as recomendações de mudança para um projeto de software. Ao comparar a acurácia do modelo de recomendação de mudanças baseado em AG com o modelo estático, gerado a partir do trabalho de Moonen et al., variando valores de suporte e confiança de acordo com valores usados na literatura [2, 29], foi possível observar que a abordagem proposta é capaz de produzir resultados semelhantes ao trabalho de Moonen et al. para projetos grandes. Entretanto, o AG conseguiu otimizar valores de suporte, confiança e tamanho de treinamento para o projeto com menor quantidade de histórico, para o qual a regressão não foi capaz de determinar o conjunto de treinamento.

Na segunda questão de pesquisa, verificamos que a inserção de novas mudanças no sistema deterioram a estabilidade dos modelos de recomendação de mudanças. Especialmente no Laravel, um dos menores projetos, a acurácia das recomendações decresceu de 0.15 para 0.05 à medida que novas mudanças eram consideradas para o teste dos modelos.

Como trabalho futuro, pretendemos utilizar nossa abordagem com o algoritmo *Targeted Association Rule Mining for All Queries* (TARMAQ) [23] e comparar com o algoritmo *Apriori*, além de estender a análise para uma maior quantidade projetos.

REFERÊNCIAS

[1] R. Agrawal, T. Imieliński, and A. Swami. 1993. Mining Association Rules Between Sets of Items in Large Databases. *SIGMOD Rec.* 22, 2 (June 1993), 207–216.

[2] G. Bavota, B. Dit, R. Oliveto, M. Di Penta, D. Poshyvanyk, and A. De Lucia. 2013. An Empirical Study on the Developers' Perception of Software Coupling. In *Proceedings of the 2013 International Conference on Software Engineering (ICSE '13)*. IEEE Press, Piscataway, NJ, USA, 692–701.

[3] G. Canfora, M. Ceccarelli, L. Cerulo, , and M. Di Penta. 2010. Using multivariate time series and association rules to detect logical change coupling: An empirical study. In *IEEE International Conference on Software Maintenance, ICSM*.

[4] F. Colares, J. Souza, R. Carmo, C. Pádua, and G. R. Mateus. 2009. A New Approach to the Software Release Planning. In *2009 XXIII Brazilian Symposium on Software Engineering*. 207–215.

[5] B. Dit, M. Wagner, S. Wen, W. Wang, M. Linares-Vásquez, D. Poshyvanyk, and H. Kagdi. 2014. ImpactMiner: A Tool for Change Impact Analysis. In *Companion Proceedings of the 36th International Conference on Software Engineering (ICSE Companion 2014)*. ACM, New York, NY, USA, 540–543.

[6] R. A. M. Filho and S. R. Vergilio. 2015. A Mutation and Multi-objective Test Data Generation Approach for Feature Testing of Software Product Lines. In *2015 29th Brazilian Symposium on Software Engineering*. 21–30.

[7] H. Gall, K. Hajek, and M. Jazayeri. 1998. Detection of logical coupling based on product release history. In *Proceedings. International Conference on Software Maintenance (Cat. No. 98CB36272)*. 190–198.

[8] S. Gotshall and B. Rylander. 2002. Optimal population size and the genetic algorithm. *Population* 100, 400 (2002), 900.

[9] T. L. Graves, A. F. Karr, J. S. Marron, and H. Siy. 2000. Predicting fault incidence using software change history. *IEEE Transactions on Software Engineering* 26, 7 (Jul 2000), 653–661.

[10] Michael Hahsler, Sudheer Chelluboina, Kurt Hornik, and Christian Buchta. 2011. The Arules R-Package Ecosystem: Analyzing Interesting Patterns from Large Transaction Data Sets. *Journal of Machine Learning Research* 12 (July 2011), 2021–2025.

[11] A. E. Hassan. 2008. The road ahead for Mining Software Repositories. In *2008 Frontiers of Software Maintenance*. 48–57.

[12] A. E. Hassan and R. C. Holt. 2004. Predicting change propagation in software systems. In *20th IEEE International Conference on Software Maintenance, 2004. Proceedings*. 284–293.

[13] K. Herzig and A. Zeller. 2013. The Impact of Tangled Code Changes. In *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR '13)*. IEEE Press, Piscataway, NJ, USA, 121–130.

[14] J. H. Holland. 1975. *Adaptation in Natural and Artificial Systems*. Vol. Ann Arbor. 183 pages. arXiv:0262082136

[15] M. M. Lehman. 1980. Programs, life cycles, and laws of software evolution. *Proc. IEEE* 68, 9 (September 1980), 1060–1076.

[16] O. Maimon and L. Rokach. 2010. *Data Mining and Knowledge Discovery Handbook* (2nd ed.). Springer Publishing Company, Incorporated.

[17] H. Malik and A.E. Hassan. 2008. Supporting software evolution using adaptive change propagation heuristics. In *IEEE International Conference on Software Maintenance, 2008. ICSM 2008*. 177–186.

[18] L. Moonen, S. Di Alesio, D. Binkley, and T. Rolfsnes. 2016. Practical Guidelines for Change Recommendation using Association Rule Mining. In *International Conference on Automated Software Engineering (ASE)*. ACM.

[19] L. Moonen, S. Di Alesio, T. Rolfsnes, and D. Binkley. 2016. Exploring the Effects of History Length and Age on Mining Software Change Impact. In *International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE.

[20] G. A. Oliva and M. A. Gerosa. 2015. Change Coupling Between Software Artifacts: Learning from Past Changes. In *The Art and Science of Analyzing Software Data*, C. Bird, T. Menzies, and T. Zimmermann (Eds.). Morgan Kaufmann, 285–324.

[21] G. Pinto, I. Steinmacher, and M. A. Gerosa. 2016. More Common Than You Think: An In-depth Study of Casual Contributors. In *IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Vol. 1. 112–123.

[22] B. Ray, D. Posnett, V. Filkov, and P. Devanbu. 2014. A Large Scale Study of Programming Languages and Code Quality in Github. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2014)*. ACM, New York, NY, USA, 155–165.

[23] T. Rolfsnes, S. Di Alesio, R. Behjati, L. Moonen, and D. W. Binkley. 2016. Generalizing the Analysis of Evolutionary Coupling for Software Change Impact Analysis. In *International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 201–212.

[24] L. Scrucca. 2013. GA: A Package for Genetic Algorithms in R. *Journal of Statistical Software* 53, 4 (2013), 1–37. http://www.jstatsoft.org/v53/i04/

[25] CS. K. Selvi and A. Tamilarasi. 2009. An automated association rule mining technique with cumulative support thresholds. *Int. J. Open Problems in Compt. Math* 2, 3 (2009).

[26] I. Steinmacher, T. Conte, M. A. Gerosa, and D. Redmiles. 2015. Social Barriers Faced by Newcomers Placing Their First Contribution in Open Source Software Projects. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work and Social Computing (CSCW '15)*. ACM, New York, NY, USA, 1379–1392.

[27] W. Su, Y. Yuan, and M. Zhu. 2015. A Relationship Between the Average Precision and the Area Under the ROC Curve. In *Proceedings of the 2015 International Conference on The Theory of Information Retrieval (ICTIR '15)*. ACM, New York, NY, USA, 349–352.

[28] Z. Zheng, R. Kohavi, and L. Mason. 2001. Real World Performance of Association Rule Algorithms. In *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '01)*. ACM, New York, NY, USA, 401–406.

[29] T. Zimmermann, P. Weißgerber, S. Diehl, and A. Zeller. 2005. Mining version histories to guide software changes. *IEEE Transactions on Software Engineering* 31, 6 (2005), 429–445.