Uma Abordagem Multiobjetiva para Seleção de Bugs

Rogério Lucas Marinho da Silva, Gledson Elias

Centro de Informática – Universidade Federal da Paraíba (UFPB)

rogeriofilho2@gmail.com, gledson@ci.ufpb.br

Abstract. *Software bugs are common and incur high costs during development and maintenance of software systems. In complex scenarios, it is possible to have a large number of unfixed bugs, representing a challenge the selection of bugs to be fixed in the next release. If manually selected, it can be a timing consuming approach, with a high level of difficulty and prone to error. In such a context, enhancing a mono-objective approach, this paper proposes an automated multi-objective approach for selecting interdependent bugs based on their priority and severity, and dependencies among software components.*

Resumo. *Defeitos de software (bugs) são comuns e acarretam elevados custos durante o desenvolvimento e a manutenção de sistemas de software. Em cenários complexos, pode haver uma grande quantidade de bugs a serem corrigidos, representando um desafio a seleção dos bugs para correção no próximo release. Se selecionados manualmente, podem consumir muito tempo, com um elevado nível de dificuldade e com tendências a erros. Neste contexto, como uma melhoria de uma proposta mono-objetiva, este artigo propõe uma abordagem automática multiobjetiva de seleção de bugs interdependentes considerando a prioridade e a severidade dos bugs, como também as dependências entre componentes de software.*

1. Introdução

A correção de bugs gera elevados custos para os sistemas de software. Ao longo do tempo, a complexidade das aplicações tende a crescer e com isso o número de bugs tende a aumentar. Grandes projetos, como o navegador Firefox (Mozilla, 2024) e a IDE Eclipse (Eclipse Foundation, 2024), possuem milhares de bugs registrados ao longo de sua história. Por exemplo, colaboradores do Projeto Eclipse reportaram mais de 3.000 bugs num período de 4 meses (Averik *et al*.,2006). Na perspectiva de gerenciamento de projetos, a tarefa de seleção dos bugs a serem corrigidos torna-se complexa e sujeita a erros. Além disso, deve-se ressaltar que os esforços de gerenciamento e desenvolvimento precisam ser divididos entre outras atividades como implementação de novas funcionalidades, integração de mudanças e reuniões gerenciais. Há, portanto, uma necessidade de simplificar e automatizar a seleção de bugs a serem corrigidos de uma forma efetiva, reduzindo o esforço e o tempo empreendidos e recomendando bons conjuntos de bugs para correção na perspectiva de satisfação do produtor do software e dos seus respectivos clientes.

Neste contexto, a abordagem multiobjetiva aqui apresentada se baseia na proposta de Lima e Elias (2017), que propõe uma abordagem automática mono-objetiva de seleção de bugs interdependentes, levando em consideração a prioridade e a severidade dos bugs,

Catalogação na publicação
Seção de Catalogação e Classificação

S586a Silva, Rogerio Lucas Marinho da.
Uma abordagem multiobjetiva para seleção de bugs / Rogerio Lucas Marinho da Silva. - João Pessoa, 2024.
17 f. : il.

Orientação: Gledson Elias da Silveira.
TCC (Graduação) - UFPB/CI.

1. Problema de seleção de bugs. 2. Busca multiobjetiva. 3. Engenharia de software baseada em buscas. 4. Algoritmos genéticos. 5. Meta-heurísticas. I. Silveira, Gledson Elias da. II. Título.

UFPB/CI CDU 004.4

Elaborado por Michelle de Kássia Fonseca Barbosa - CRB-738

indicadas pelo produtor e respectivos clientes, além de analisar as dependências entre bugs com base nas dependências entre componentes arquiteturais. Na proposta mono-objetiva original, Lima e Elias (2017) adotam a metaheurística evolucionária clássica de algoritmos genéticos para representação, modelagem e resolução do problema de seleção de bugs, tendo o orçamento disponível como uma restrição que limita as possíveis soluções recomendadas. Na melhoria aqui proposta, a modelagem e a resolução do problema de seleção bugs são adaptadas para um problema multiobjetivo, passando a adotar uma metaheurística evolucionária multiobjetiva, também baseada em algoritmos genéticos, denominada NSGA-II (Deb *et al.*, 2002), permitindo a identificação e análise de um conjunto de soluções não-dominadas, que proporcionam um compromisso adequado entre todos os objetivos sem degradar nenhum deles. Nos experimentos com 20 e 50 bugs, os resultados mostram uma boa aproximação com a fronteira de Pareto, que corresponde ao conjunto conhecido ou de referência de soluções ótimas não-dominadas.

O restante do artigo está estruturado da seguinte maneira. A seguir, os trabalhos relacionados são discutidos, incluindo a proposta original de Lima e Elias (2017), pontuando diferenças com a abordagem aqui proposta. A Seção 3 introduz conceitos e fundamentos da Engenharia de Software Baseada em Buscas, incluindo metaheurísticas evolucionárias e indicadores de qualidade, essenciais ao entendimento e desenvolvimento da abordagem proposta. Na Seção 4, a abordagem multiobjetiva proposta é apresentada, evidenciando as adaptações da proposta mono-objetiva original. A Seção 5 objetiva descrever e discutir os resultados encontrados. Por fim, na conclusão, reiteram-se os pontos fortes e os pontos de melhoria para trabalhos futuros.

2. Trabalhos Relacionados

Como já mencionado, (Lima e Elias, 2017) representa o principal trabalho relacionado à proposta apresentada e avaliada neste artigo. A solução proposta pelos autores é uma abordagem automática de seleção de bugs baseada na arquitetura de componentes e considerando as dependências entre bugs. Inicialmente, considerando as dependências entre componentes arquiteturais, bem como a indicação da ocorrência de bugs nestes componentes, é possível estimar as dependências entre bugs. Para tal, os autores assumem que bugs em componentes dependentes também apresentam a mesma relação de dependência entre si. Em seguida, considerando a prioridade e a severidade dos bugs, reportadas pelo produtor e respectivos clientes, pode-se estimar o nível de prioridade e o nível de severidade de uma dada solução de seleção de bugs. Por fim, considerando a função objetivo definida pela média ponderada dos níveis de prioridade e severidade, a terceira etapa adota algoritmos genéticos para recomendar a melhor solução encontrada de seleção de bugs. Como principal limitação do trabalho, a formulação mono-objetiva foca apenas na melhor solução encontrada, diminuindo o número de escolhas e nuances para a tomada de decisão, não proporcionando um compromisso adequado entre os níveis de prioridade e severidade das soluções não-dominadas.

De forma similar a abordagem aqui proposta, Dreyton (2016) utiliza uma abordagem multiobjetiva para atacar o problema de priorização de bugs em repositórios de código aberto, considerando informações de severidade, prioridade, relevância e nível de experiência dos desenvolvedores em certas áreas do projeto, fornecidas pela própria comunidade, referências técnicas ou especialistas de domínio. Neste caso, as funções objetivo buscam maximizar a importância e minimizar os riscos das soluções encontradas, recomendando bugs para correção com base no tipo de bug e na experiência

dos desenvolvedores nas áreas do projeto. Contudo, como limitação, o número de bugs a serem priorizados é definido como dado de entrada. Diferentemente, na abordagem aqui proposta, a quantidade de bugs não é predefinida, mas avaliada de forma automática e indiretamente restringida pelo orçamento disponível no projeto. Logo, a presente abordagem é mais prática e reduz o esforço, pois não requer a antecipação manual de quantos bugs devem ser corrigidos.

Por fim, a proposta que mais se assemelha à aqui apresentada é a abordagem de Almhana *et al.* (2020), que também recomenda um conjunto de bugs para correção de forma automatizada, levando em consideração as dependências entre bugs e a ordem de correção. Os autores entendem dependências entre bugs apenas quando os bugs ocorrem nos mesmos arquivos de classe. Neste caso, as funções objetivo buscam minimizar a quantidade de classes, que serão alvo da correção do conjunto de bugs recomendados, e maximizar o nível de prioridade dos bugs escolhidos, estimado a partir de suas prioridades individuais nos relatórios do BTS (*Bug Tracking System*). Ou seja, quanto maior a prioridade dada a um bug em seu relatório, maior sua prioridade de correção no conjunto recomendado. Diferente da abordagem aqui proposta, percebe-se que o autor não leva em conta múltiplos clientes, com diferentes prioridades, severidades e grau de importância.

3. Fundamentos

A Engenharia de Software baseada em Buscas, do inglês, *Search-Based Software Engineering* (SBSE), é uma área de estudo relativamente recente na qual problemas da engenharia de software são reformulados, modelados e resolvidos como problemas de otimização, adotando diferentes classes de metaheurísticas, tais como: algoritmos genéticos, arrefecimento simulado, colônia de formigas e enxame de partículas. Em todo ciclo de vida dos projetos de software, as estratégias da SBSE podem e estão sendo aplicadas (Harman *et al.*, 2012), incluindo engenharia de requisitos, projeto de software, teste de software, gerenciamento e planejamento de projetos de software, etc.

Inicialmente, a SBSE adotava apenas metaheurísticas mono-objetivas para resolver problemas da engenharia de software, que, em geral, ponderava diferentes variáveis de decisão em uma única função objetivo para recomendar uma única solução próxima da ótima. Com o passar do tempo, novos desafios necessitaram de novos métodos de resolução. Nos últimos anos, há uma maior preocupação com problemas multiobjetivos, ou seja, problemas nos quais existe a necessidade de otimização de duas ou mais variáveis de decisão em geral conflitantes, permitindo a identificação e análise de um conjunto de soluções não-dominadas. Logo, não mais existe uma única solução próxima da ótima, mas um conjunto de soluções não-dominadas que proporcionam um compromisso ou balanceamento adequado entre todos os objetivos sem degradar nenhum deles. Dito de outra forma, soluções não-dominadas são necessariamente as melhores em todos os objetivos, tendendo a possuir prós e contras entre seus objetivos, não sendo possível melhorar um objetivo sem afetar negativamente outro. Portanto, fica a cargo de um ser humano tomar a decisão final baseada em um conjunto de possíveis soluções.

Em um problema multiobjetivo de minimização de n variáveis de decisão, uma solução candidata é definida por um vetor $\vec{x} = [x_1, x_2, \dots, x_n]^T$, que satisfaça as restrições impostas e minimize o vetor de funções objetivo $\vec{f}(\vec{x}) = [f_1(\vec{x}), f_2(\vec{x}), \dots, f_k(\vec{x})]^T$. Por analogia, podemos entender problemas de maximização como problemas de minimização de funções negativas.

Pela natureza dos problemas multiobjetivos, faz-se necessário adotar um critério para ranqueamento das soluções. Para tal, adota-se o conceito de dominância que define a melhor dentre duas soluções. Segundo Durillo *et al*. (2010), em um problema de minimização, uma solução $\vec{x}' = [x_1', x_2', \dots, x_n']$ domina outra solução $\vec{x}'' = [x_1'', x_2'', \dots, x_n'']$, se e somente se $f_i(\vec{x}') \leq f_i(\vec{x}''), \forall\, i = 1, 2, \dots, k$. Além disso, deve existir pelo menos um objetivo j $(1 \leq j \leq k)$ tal que $f_j(\vec{x}') < f_j(\vec{x}'')$. Para casos de maximização, basta trocar o sinal de $\leq$ para $\geq$, e o sinal de $<$ para $>$.

Outro conceito primordial é a *fronteira de Pareto*, que se refere ao conjunto de soluções que não são dominadas por nenhuma outra solução em toda a região do espaço de busca. É importante ressaltar que a busca por soluções próximas do conjunto ótimo de Pareto é o foco principal de um problema de otimização multiobjetivo.

3.1. Indicadores de Qualidade

Infelizmente, em função do tamanho do espaço de busca ou mesmo do desempenho da metaheurística multiobjetiva, nem sempre é possível encontrar a fronteira ótima de Pareto. Nesse sentido, existem diferentes indicadores que permitem avaliar a qualidade das soluções não-dominadas encontradas.

É crítico que as melhores soluções encontradas tenham valores das funções objetivos com uma pequena ou nenhuma diferença para a fronteira de Pareto. Segundo, é importante que as soluções sejam bem distribuídas ou espalhadas ao longo da fronteira, pois indica que houve exploração homogênea do espaço de busca, sem deixar áreas inexploradas. A Figura 1 mostra os seguintes casos: *a)* soluções próximas da fronteira, mas sem uniformidade no espalhamento; b) soluções afastadas da fronteira, mas com espalhamento uniforme; c) soluções próximas da fronteira e com espalhamento uniforme.

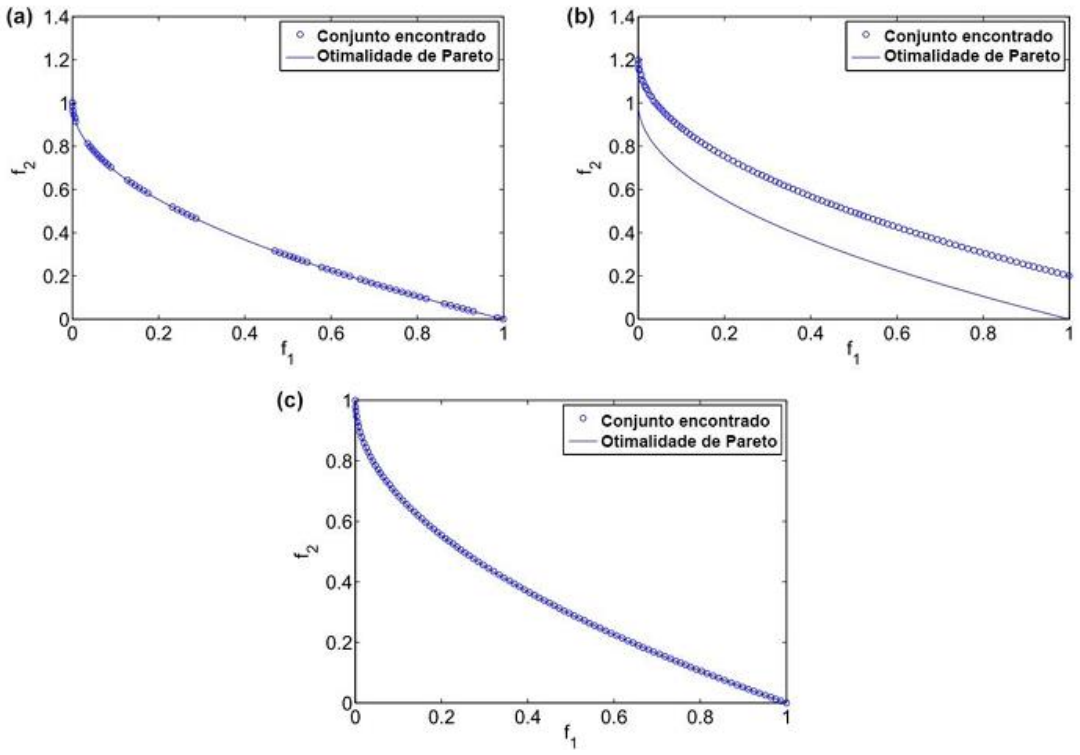

Figura 1. Representação de proximidade e espalhamento (apud Lima, 2019)

No contexto de metaheurísticas multiobjetivas, o indicador de espalhamento $\Delta = (d_f + d_l + \sum_{i=1}^{N-1}|d_i - \bar{d}|)/(d_f + d_l + (N-1).\bar{d})$ diz respeito ao nível de diversidade do conjunto de soluções encontradas, demonstrando a extensão do espalhamento na fronteira. A Figura 1a é um exemplo ruim de espalhamento, enquanto a Figura 1b e Figura 1c representam bons exemplos de espalhamento. Considerando a Figura 2, os termos d_f e d_l são as distâncias euclidianas entre os pontos extremos da fronteira de Pareto e os extremos da fronteira encontrada; termo d_i é a distância entre os pontos consecutivos da fronteira encontrada; e $\bar{d}$ é a distância média entre pontos consecutivos da fronteira encontrada. Logo, o melhor resultado possível para o espalhamento é valor 0, que indicaria que os pontos encontrados estão uniformemente distribuídos pela fronteira de Pareto.

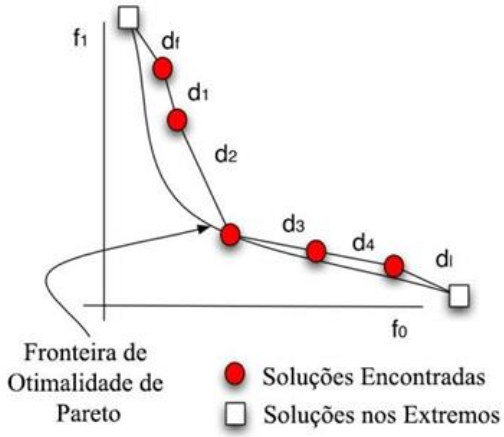

Figura 2. Espalhamento para dois objetivos (apud Lima, 2019)

Outro importante indicador de qualidade é o hipervolume $HV = \bigcup_{i-1}^{|Q|} v_i$, que, além de indicar a convergência ou proximidade do conjunto de soluções não-dominadas Q em relação à fronteira de Pareto, também considera o espalhamento das soluções encontradas. Como ilustrado na Figura 3, o hipervolume é representado pela união dos hipercubos v_i formados por um ponto de referência W e cada solução encontrada. Uma forma de definir o ponto de referência W é construir um vetor com os piores valores dos vários objetivos nas soluções Q e na fronteira de Pareto.

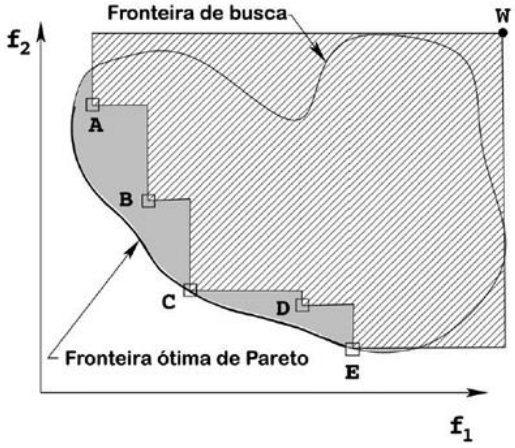

Figura 3. Hipervolume para dois objetivos (apud Lima, 2019)

O hipervolume de um conjunto de soluções Q pode ser um indicador que à primeira vista não mostra tão claramente a noção de proximidade em relação à fronteira de Pareto. Então, em geral, adota-se o hipervolume normalizado $HVR = HV(Q)/HV(P^*)$, cujo valor varia entre 0 e 1. Observe que o HVR representa a relação entre o hipervolume das soluções encontradas Q e o hipervolume da fronteira de referência P^*, que pode ser a fronteira de Pareto real, quando conhecida, ou apenas uma fronteira aproximada, obtida por exemplo a partir de inúmeras execuções sucessivas de metaheurísticas multiobjetivas.

3.2. Metaheurísticas

Para garantir os melhores resultados possíveis, ditos ótimos, para um problema de otimização, seria necessário um algoritmo que garante encontrar a melhor solução. Contudo tais algoritmos tendem a necessitar de um tempo inviável para problemas com espaços de busca que crescem exponencialmente com o aumento dos valores possíveis de entrada. Em oposição, metaheurísticas são algoritmos, normalmente estocásticos, que procuram soluções aceitáveis em um tempo razoável. Tais estratégias recorrem a ideias intuitivas (heurísticas) para investigar o espaço de busca atrás de boas soluções com custo computacional aceitável, mas sem a garantia de encontrar a solução ótima (Chopard e Tomassini, 2018). Um dos algoritmos pertencentes às metaheurísticas são os algoritmos genéticos, que será detalhado a seguir.

3.2.1. Algoritmos Genéticos

Algoritmos genéticos são uma categoria de algoritmos inspirados no processo de seleção natural, descrito por Charles Darwin. Neste processo, a partir de uma população inicial, os indivíduos passam pelas etapas de seleção, cruzamento e mutação. Ao longo de múltiplas gerações, as soluções de maior qualidade são progressivamente geradas e perpetuadas para iterações subsequentes. Este processo persiste até que um critério de parada predeterminado seja satisfeito. Uma analogia comum é estabelecida entre as soluções possíveis e os indivíduos de uma população, cuja adaptabilidade ao ambiente é representada pelo valor de suas respectivas funções objetivo. Podemos pensar a solução como um conjunto de genes, que indicam características presentes ou não. Tais soluções podem se reproduzir gerando novas soluções e passar por um processo de variabilidade genética que modifica as soluções. E, principalmente, usando um mecanismo de memória, soluções melhores são escolhidas e mantidas (Chopard e Tomassini, 2018).

O primeiro passo é a criação da população inicial, que em geral é realizada de forma aleatória. O tamanho da população é bem pequeno em relação ao número total de soluções possíveis. Em seguida, indivíduos são selecionados com base no seu nível de adaptabilidade, onde indivíduos mais preparados têm mais chances por terem melhores valores de funções objetivo. Por exemplo, na seleção por roleta, a probabilidade de cada indivíduo ser escolhido é definida pela relação entre o valor da sua função objetivo e o somatório dos valores das funções objetivo de todos os indivíduos da população.

O segundo passo é o cruzamento de duas soluções pais, escolhidas para a geração de mais duas novas soluções filhas. Usualmente, com alta probabilidade, as soluções filhas são diferentes das soluções pais. Por exemplo, no cruzamento em um ponto, é selecionado algum ponto aleatoriamente e então os genes à esquerda do primeiro pai e à direita do segundo pai vão para o primeiro filho, e vice-versa para o segundo filho.

Após o cruzamento, segue-se com a mutação com o objetivo de trazer mais diversidade às soluções. A mutação consiste em gerar alguma modificação nas soluções filhas, seja inserindo ou retirando uma ou mesmo várias de suas características. A probabilidade da mutação é pequena e tende a ser definida por $1/n$, onde o termo n representa o número de genes de um indivíduo.

Ao final da iteração, gera-se uma nova população, em geral, mantendo boas soluções. Há várias técnicas que podem ser adotadas, como por exemplo, simplesmente substituir toda a população antiga pelas soluções filhas gerados, ou qualitativamente escolher as melhores entre as soluções pais e filhas.

Os processos de seleção, cruzamento, mutação e substituição são realizados enquanto um critério de parada predefinido não é atingido. Critérios comuns são o número de gerações, os valores objetivos não crescem depois de determinado número de gerações, ou uma solução satisfatória é encontrada (Chopard e Tomassini, 2018).

3.2.2. NSGA-II

Originalmente, os algoritmos genéticos foram pensados para problemas mono-objetivos. Em decorrência da necessidade de resolução de problemas multiobjetivos, houve a criação e posterior evolução de novos algoritmos, como por exemplo o NSGA (*Non-dominated Sorting Genetic Algorithm*) (Srinivas e Deb, 1995), e sua versão melhorada NSGA-II (Deb *et al.*, 2002), que será descrito e utilizado neste trabalho, pois resolve eficientemente problemas multiobjetivos de até três objetivos (Ma *et al.*, 2023).

O NSGA-II possui o seguinte funcionamento: *(i)* cria a população inicial aleatoriamente; *(ii)* realiza a atribuição de dominância das soluções de acordo com a regra de dominância de Pareto; *(iii)* separa as soluções em fronteiras, onde as soluções não-dominadas ficam na primeira fronteira, as soluções por elas dominadas ficam na segunda fronteira e assim por diante; e *(iv)* realiza a classificação de espalhamento, onde cada solução recebe um valor de distância em relação aos seus vizinhos, de modo que aquelas com maior de distância são melhor avaliadas.

Na atribuição de dominância, para cada solução p da população P, é definido um valor n_p que indica o número de soluções que dominam p. Cria-se também um conjunto S_p, que representa as soluções dominadas por p. Esse passo possui complexidade $O(MN^2)$, onde M é o número de objetivos e N é o número de soluções. Em seguida, na separação em fronteiras, as soluções que possuem o valor n_p como 0, fazem parte da primeira fronteira. Então, as soluções dominadas pela primeira fronteira têm seus valores de n_p reduzido em 1. Caso resulte em 0, tais soluções fazem parte da segunda fronteira. O processo se repete até que todas as soluções estejam em pelo menos uma fronteira.

Na classificação de espalhamento, ordena-se a população pelo valor das funções objetivo de forma crescente. Em seguida, a primeira e a última soluções recebem valor de distância infinita. As demais soluções recebem valor igual a diferença absoluta e normalizada dos valores das funções objetivo das soluções adjacentes. O valor final será a soma das distâncias calculadas para cada objetivo. Esse passo possui complexidade $O(MN \log N)$, onde M é o número de objetivos e N o tamanho da população. Note que o espalhamento somente é calculado para elementos de uma mesma fronteira, pois o elitismo do algoritmo escolhe as melhores fronteiras para as próximas gerações.

Também é necessário descrever o operador de comparação, utilizado para substituir a população a cada iteração. Considerando diversas soluções, as melhores são aquelas pertencentes às menores fronteiras, ou, se estiverem na mesma fronteira, aquelas que tiverem as maiores distâncias de espalhamento. Observe que, na fronteira que não cabe por completo na nova população, apenas as soluções com maiores distâncias de espalhamento são selecionadas.

A Figura 4 descreve o ciclo de execução do NSGA-II, representado pela união do conjunto de soluções pais e filhas, atribuição de dominância, cálculo da distância de espalhamento e inclusão do máximo de fronteiras para a nova população, inserindo prioritariamente as soluções das primeiras fronteiras e complementando com as soluções com maiores distâncias de espalhamento na última fronteira incluída. Note que, ao final da iteração, os operadores de seleção, cruzamento e mutação são executados para geração de uma nova população. Esse ciclo acontece pelo número predefinido de gerações N.

```
01 R_t = P_t ∪ Q_t          //criar um conjunto único de pais e filhos de tamanho 2N
02 F =fast-non-dominated-sort(R_t)          //atribuição de dominância
03 P_{t+1} = ∅ and i = 1
04 until |P_{t+1}| + |F_i| ≤ N    //incluir fronts na nova população de pais até que o tamanho do fronts selecionados até o momento exceda o tamanho N da população
05     Crowding-distance-assignment(F_i)    //cálculo de distância
06     P_{t+1} = P_{t+1} ∪ F_i    //incluir o front na nova população de pais
07     i = i + 1          //incrementar e testar o próximo front i+1
08 Sort(F_i, ≺_n)        //ordenar os elementos do último front i a fim de preencher exatos N elementos na população de pais
09     P_{t+1} = P_{t+1} ∪ F_i[1:(N − |P_{t+1}|)]          //escolha os primeiros (N-|P_{t+1}|) elementos de F_i para preencher o restante da solução
10 Q_{t+1} =make-new-pop(P_{t+1})          //utilizar os operadores de seleção, cruzamento e mutação para gerar a nova população Q_{t+1}
11 t = t + 1    //incrementar o contador de geração
```

Figura 4. Pseudocódigo do ciclo principal do NSGA-II (apud Amaral, 2017)

4. Abordagem Proposta

O objetivo desta seção é apresentar a abordagem multiobjetivo aqui proposta, que, como já mencionado, é uma adaptação da modelagem matemática e posterior mudança da estratégia de resolução do Problema de Seleção de Bugs (*Bug Selection Problem* - BSP) da abordagem mono-objetiva apresentada em (Lima e Elias, 2017). Como originalmente definido pelos autores, o problema BSP foca na necessidade de escolher o conjunto de bugs que devem ser corrigidos como parte da próxima versão do software. Lima e Elias (2017) afirmam que o BSP é um problema NP-Hard, que deve então ser solucionado com técnicas de SBSE. Neste contexto, a seguir, a abordagem mono-objetiva original será introduzida, evidenciando onde pertinente a adaptação matemática para o problema BSP multiobjetivo.

De forma similar à abordagem proposta por Lima e Elias (2017), a abordagem multiobjetiva aqui proposta é também estruturada em três estágios: *(i)* ***Recomendação das Dependências dos Bugs*** – avalia as dependências dos componentes na arquitetura do sistema de software, estabelecendo relações de precedência e concorrência entre as respectivas tarefas de correção de bugs; *(ii)* ***Recomendação das Prioridades e Severidades dos Bugs*** – avalia a prioridade e severidade dos bugs na perspectiva do produtor de software e seus vários clientes, adotando um modelo multicliente que melhor calibra a importância e o impacto dos bugs relatados; e *(iii)* ***Recomendação das Tarefas de Correção de Bugs*** – adota uma metaheurística multiobjetiva para indicar as tarefas mais relevantes de resolução de bugs, maximizando os níveis de prioridade de severidade das soluções recomendadas, mas limitadas ao orçamento predefinido do projeto.

É importante ressaltar que, no terceiro estágio, diferente da abordagem de Lima e Elias (2017), que adota uma única função objetivo e uma metaheurística mono-objetiva, a abordagem aqui proposta adota duas funções objetivo, relacionadas aos níveis de prioridade e severidade das soluções, bem como explora uma metaheurística multiobjetivo, especificamente o algoritmo NSGA-II.

4.1. Recomendação das Dependências dos Bugs

Como proposto por Lima e Elias (2017), as dependências entre componentes de software implicam em dependências entre seus respectivos bugs. Portanto, avaliando a arquitetura baseada em componentes do projeto de software é possível derivar as dependências entre bugs. Nesta direção, considerando o conjunto de bugs $B = \{b_1, b_2, \dots, b_m\}$ e o conjunto de componentes arquiteturais $A = \{a_1, a_2, \dots, a_k\}$, a partir dos relatórios de bugs no BTS, que indicam os componentes onde ocorreram os bugs, é possível realizar o *mapeamento de bugs para componentes*, representado pela matriz binária $BA_{B\times A}$, definida em (1), onde linhas e colunas representam bugs e componentes, respectivamente. É importante destacar que um bug ocorre em apenas um componente, mas um componente pode ter vários bugs. Outra informação primordial existente nos relatórios de bugs no BTS é o custo para correção dos bugs, dado pelo conjunto $V = \{v_i \mid \exists_{b_i}\, b_i \in B \ \wedge \ v_i > 0\}$.

$$BA_{B\times A} = \left\{ba_{i,j} \;\middle|\; \exists_{b_i}\, b_i \in B \ \wedge\ \exists_{a_j}\, a_j \in A \ \wedge\ ba_{i,j} \in \{0,1\}\right\}$$
$$ba_{i,j} = \begin{cases} 1 & se\ bug\ b_i\ ocorre\ no\ componente\ c_j \\ 0 & caso\ contrário \end{cases} \tag{1}$$

Agora, considerando a arquitetura do sistema de software, que pode ser vista como um grafo direcionado, onde vértices são componentes e arestas direcionadas representam as relações entre interfaces requeridas e providas pelos componentes, é possível extrair as *dependências de componentes*, representadas pela matriz quadrada binária $AD_{A\times A}$, definida em (2), onde linhas e colunas representam componentes, indicando que o componente a_j na linha j depende de outro componente a_k na linha k se e somente se a_j requer uma operação provida por a_k. Logo, o termo $ad_{j,k}$ recebe o valor 1 quando o componente a_j depende do componente a_k. Caso contrário, $ad_{j,k}$ recebe o valor 0. Note que um componente pode depender de zero ou mais componentes.

$$AD_{A\times A} = \left\{ad_{j,k} \;\middle|\; \exists_{a_j}\, a_j \in A \ \wedge\ \exists_{a_k}\, a_k \in A \ \wedge\ ad_{j,k} \in \{0,1\}\right\}$$
$$ad_{j,k} = \begin{cases} 1 & se\ componente\ a_j\ depende\ do\ componente\ a_k \\ 0 & caso\ contrário \end{cases} \tag{2}$$

O passo seguinte é identificar as *dependências entre bugs*, representadas pela matriz quadrada binária $BD_{B\times B}$, definida em (3). Para tal, operações de matrizes booleanas são realizadas levando em consideração a ocorrência dos bugs nos componentes $BA_{B\times A}$ e as dependências entre componentes $AD_{A\times A}$. Na extração das dependências entre bugs, explorando o conceito de fecho transitivo do grafo direcionado definido pela arquitetura do sistema, todas as dependências diretas e indiretas entre bugs são capturadas. Neste processo, um novo grafo direcionado é definido, contendo o mesmo conjunto de vértices, mas com novas arestas entre os componentes a_x e a_y, se e somente se, a_y for alcançável a partir de a_x no grafo original.

$$BD_{B\times B} = \left(BA_{B\times A} \cdot \sum_{n=0}^{|A|} AD_{A\times A}^{n} \right) \cdot BA_{B\times A}^{T} \tag{3}$$

Inicialmente, deve-se calcular cada n-ésima potência da matriz de alcançabilidade (Warfield, 1973), representada por $AD_{A\times A}^{n}$, onde $AD_{A\times A}^{0}$ é a matriz identidade, encontrando as dependências diretas e indiretas que cada componente possui com outros para todos os caminhos de tamanhos sucessivos. Em seguida, a matriz de visibilidade $AD_{A\times A}^{VD}$, resultante do somatório das matrizes $AD_{A\times A}^{n}$, indica todas as dependências diretas e indiretas entre componentes para todos os tamanhos de caminhos possíveis. Posteriormente, multiplicando o mapeamento de bugs para componentes $BA_{B\times A}$ pela matriz de visibilidade $AD_{A\times A}^{VD}$ obtém-se a matriz de dependências $BA_{B\times A}^{VD}$ contendo todas as dependências diretas e indiretas de bugs para componentes. Por fim, multiplicando a matriz de dependências $BA_{B\times A}^{VD}$ pela transposta do mapeamento de bugs para componentes $BA_{B\times A}^{T}$, gera-se a matriz de dependências entre bugs $BD_{B\times B}$, que indica todas as dependências diretas e indiretas entre bugs.

Observe que, como definido em (Lima e Elias, 2017), a matriz $BD_{B\times B}$ pode ser vista como soluções básicas $BX_{B\times B}$, ou seja, $BX_{B\times B} = BD_{B\times B}$, pois para um dado bug b_i na linha i, todos os bugs b_j marcados com 1 na linha i e colunas j, definido pela matriz linha $[bx_{i,1}, bx_{i,2}, \ldots bx_{i,m}]$, representam bugs que precisam ser selecionados e corrigidos juntos com o bug b_i. Logo, caso um bug b_i seja selecionado para correção, todos os demais bugs que o bug b_i depende também precisam ser selecionados.

4.2. Recomendação das Prioridades e Severidades dos Bugs

Para estimar os níveis de prioridades e severidades dos bugs, assim como em (Lima e Elias, 2017), a abordagem aqui proposta assume que o produtor e seus respectivos clientes $C = \{c_1, c_2, \ldots, c_S\}$ proveem informações de prioridade e severidade para cada bug reportado no BTS. Para tal, assume-se que as categorias adotadas de prioridade e severidade são as mesmas definidas pela ferramenta Bugzilla (ZBC, 2024), mapeadas pela abordagem proposta em uma escala de valores, conforme mostrado na Tabela 1.

Tabela 1. Níveis e valores de prioridade e severidade

Prioridade (*PV*)	*P1*	*P2*	*P3*	*P4*	*P5*
Severidade (*SV*)	*Blocker*	*Critical*	*Major*	*Minor*	*Trivial*
Escala de Valores	1.0	0.8	0.6	0.4	0.2

Pelo lado do produtor, as prioridades e as severidades dos bugs reportados são extraídas diretamente do BTS, sendo representadas por dois conjuntos, referenciados

como *prioridade do produtor* $PP = \{pp_i \mid \exists_{b_i}\, b_i \in B \wedge pp_i \in PV\}$ e *severidade do produtor* $PS = \{ps_i \mid \exists_{b_i}\, b_i \in B \wedge ps_i \in SV\}$, respectivamente.

Pelo lado dos clientes, de forma bastante similar, as prioridades e as severidades dos bugs reportados no BTS também são representadas por dois conjuntos, referenciados como *prioridade dos clientes* $CP = \{cp_i \mid \exists_{b_i}\, b_i \in B \wedge cp_i \in [0,1]\}$ e *severidade dos clientes* $CS = \{cs_i \mid \exists_{b_i}\, b_i \in B \wedge cs_i \in [0,1]\}$, respectivamente. No entanto, como clientes podem ter interesses conflitantes sobre a seleção de bugs para correção, CP e CS não são extraídos diretamente do BTS, mas estimados de forma balanceada com base na *importância dos clientes* $I = \{\, i_x \mid \exists_{c_x}\, c_x \in C \wedge i_x \in (0,1] \,\}$, que é definida pelo produtor de acordo com seu modelo de negócio.

Portanto, conforme definido em (4) e (5), para cada bug b_i, a prioridade dos clientes cp_i e a severidade dos clientes cs_i são estimadas pela média ponderada entre a importância i_x de cada cliente c_x e, respectivamente, a prioridade específica do cliente $csp_{x,i}$ e a severidade específica do cliente $css_{x,i}$, que são extraídos diretamente do BTS, resultando em valores no intervalo $[0,1] \in \mathbb{R}$.

$$cp_i = \frac{\sum_{c_x \in C}\, i_x \cdot csp_{x,i}}{|C|} \tag{4}$$

$$cs_i = \frac{\sum_{c_x \in C}\, i_x \cdot css_{x,i}}{|C|} \tag{5}$$

Como mencionado, as prioridades e as severidades dos bugs, específicas de cada cliente, são extraídas diretamente do BTS, sendo também representadas na abordagem proposta por dois conjuntos, referenciados como *prioridade específica do cliente* $CSP_{C\times B} = \{csp_{x,i} \mid \exists_{c_x}\, c_x \in C \wedge \exists_{b_i}\, b_i \in B \wedge csp_{x,i} \in PV\}$ e *severidade específica do cliente* $CSS_{C\times B} = \{css_{x,i} \mid \exists_{c_x}\, c_x \in C \wedge \exists_{b_i}\, b_i \in B \wedge css_{x,i} \in SV\}$, respectivamente, onde PV e SV podem assumir os valores indicados na Tabela 1.

4.3. Recomendação das Tarefas de Correção dos Bugs

No terceiro estágio, como já indicado, a abordagem original de Lima e Elias (2017) adota uma única função objetivo e uma metaheurística mono-objetiva baseada em algoritmos genéticos para modelar e resolver o Problema de Seleção de Bugs como um problema de otimização, recomendando uma solução satisfatória ou até mesmo ótima. Diferentemente, a abordagem aqui proposta adota duas funções objetivo, relacionadas aos níveis de prioridade e severidade das soluções, bem como explora uma metaheurística multiobjetivo, especificamente o algoritmo NSGA-II, recomendando um conjunto de soluções não-dominadas, que proporcionam um compromisso adequado entre os objetivos sem degradar nenhum deles.

4.3.1. Geração de Soluções Candidatas

Inicialmente, para cada solução candidata, os bugs são escolhidos aleatoriamente sem considerar as dependências entre si, sendo denominada de *solução independente*, que é representada pelo cromossomo $IX = \{ix_i \mid \exists_{b_i}\, b_i \in B \wedge ix_i \in \{0,1\}\}$, onde cada gene ix_i é 1 ou 0, indicando que o bug b_i foi selecionado ou não, respectivamente.

Em seguida, a solução independente deve ser convertida em uma *solução dependente*, que agrega as restrições de dependências entre bugs, sendo representada pelo

cromossomo $X = \{x_i \mid \exists_{b_i}\, b_i \in B \ \wedge\ x_i \in \{0,1\}\}$, onde cada gene x_i é 1 ou 0, indicando que o bug b_i foi selecionado ou não, respectivamente. Por exemplo, caso um bug b_i dependa de um bug b_j, se b_i for escolhido na solução independente IX, então b_j deve ser necessariamente incluído na respectiva solução dependente X.

Para converter uma solução independente IX na respectiva solução dependente X, como adotado em Lima e Elias (2017) basta realizar a simples multiplicação de matrizes booleanas entre a solução independente IX e o conjunto de soluções básicas $BX_{B\times B}$. Matematicamente, a conversão é representada por $X = IX \cdot BX_{B\times B}$.

Como uma solução candidata não pode ultrapassar o orçamento predefinido, para cada solução dependente X, deve-se calcular o seu respectivo *custo de correção de bugs* $VS_X = \sum_{b_i \in B} v_i \cdot x_i$, calculado pelo somatório dos custos específicos v_i para cada bug selecionado x_i na solução dependente X.

4.3.2. Avaliação de Soluções Candidatas

Na abordagem mono-objetiva de Lima e Elias (2017), a função objetivo é definida pela média ponderada entre o *nível de prioridade* P_X e o *nível de severidade* S_X de cada solução. De modo a adaptar a função objetivo original para um problema multiobjetivo, a abordagem aqui proposta simplesmente adota P_X e S_X como as duas funções objetivo, permitindo a recomendação de um conjunto de soluções não-dominadas.

Como definida em (6), a função objetivo P_X favorece a seleção de soluções com o maior número de bugs de maior prioridade e com o *maior grau de cobertura* CD_X, que indica a porcentagem de bugs escolhidos na solução em relação à quantidade total de bugs $|B|$, como definido em (7). Já para avaliar a prioridade de uma solução X, indicada pela expressão no lado direito de (6), adota-se a média ponderada da prioridade do produtor pp_i e das prioridades dos clientes cp_i para cada bug b_i selecionado. Portanto, se duas soluções possuem a mesma média ponderada de prioridade, aquela com o maior grau de cobertura é favorecida pois corrige um maior número de bugs.

$$P_X = CD_X \cdot \frac{wp_p \cdot \sum_{b_i \in B} pp_i \cdot x_i + wp_c \cdot \sum_{b_i \in B} cp_i \cdot x_i}{|B|} \tag{6}$$

$$CD_X = \frac{\sum_{b_i \in B} x_i}{|B|} \tag{7}$$

Como definida em (8), a função objetivo S_X favorece a seleção do maior número possível de bugs de maior severidade e com o *maior grau de interdependência* ID_X, que representa a proporção de caminhos cobertos pela solução em relação ao total de caminhos do grafo direcionado definido pelas dependências diretas e indiretas dos bugs nas soluções básicas $BX_{B\times B}$, como definido em (9). Já para avaliar a severidade de uma solução X, indicada pela expressão no lado direito de (8), adota-se a média ponderada da severidade do produtor ps_i e das severidades dos clientes cs_i para cada bug b_i selecionado. Portanto, se duas soluções possuem a mesma média ponderada de severidade, aquela com o maior grau de interdependência é favorecida pois corrige bugs mais críticos.

$$S_X = ID_X \cdot \frac{ws_p \cdot \sum_{b_i \in B} ps_i \cdot x_i + ws_c \cdot \sum_{b_i \in B} cs_i \cdot x_i}{|B|} \tag{8}$$

$$ID_X = \frac{\sum_{b_i \in B} \sum_{b_j \in B} bx_{i,j} \cdot x_i}{\sum_{b_i \in B} \sum_{b_j \in B} bx_{i,j}} \tag{9}$$

É importante mencionar que as funções objetivo P_X e S_X podem assumir valores no intervalo $[0,1] \in \mathbb{R}$. Além disso, os termos wp e ws representam pesos normalizados, ou seja, $(wp_p + wp_c = 1)$ e $(ws_p + ws_c = 1)$, aplicados para representar a contribuição efetiva do produtor e dos respectivos clientes no nível de prioridade e severidade.

Por fim, de forma similar à proposta original de Lima e Elias (2017), caso o custo de correção dos bugs VS_X de uma dada solução X for maior que o orçamento do projeto PB, ou seja, $VS_X > P$, a abordagem proposta também penaliza a solução atribuindo o valor 0 para ambas as funções objetivo P_X e S_X, tornando-as de péssima qualidade, pois são dominadas por todas as demais soluções com $VS_X \leq P$, recomendando assim apenas soluções não-dominadas que não excedem o orçamento.

5. Resultados e Discussão

A abordagem proposta foi avaliada usando o mesmo conjunto de dados reais adotados por Lima e Elias (2017). O conjunto de dados provém do *Sistema Integrado de Gestão de Atividades Acadêmicas* (SIGAA-GRADUAÇÃO) da UFPB, que possui mais de 200 componentes de software, além de um conjunto de 50 bugs reportados, definindo um espaço de busca de 2^{50} soluções candidatas. Os relatórios de bugs foram registrados pela equipe de desenvolvimento e testes da plataforma e por três coordenadores de curso, considerados como clientes. Por fim, é importante mencionar que o orçamento do projeto foi analisado em porcentagens do custo total dos bugs, a saber: 5, 10, 20, 25, 30, 40, 50, 60, 70, 75, 80, 90, 95 e 100%.

Como dito, a abordagem aqui proposta se diferencia de Lima e Elias (2017) por modelar o problema como multiobjetivo e utilizar uma metaheurística adequada, neste caso o algoritmo NSGA-II (Deb *et al.*, 2002). Para conduzir os experimentos de avaliação, reusando uma implementação já amplamente testada do NSGA-II, utilizou-se o *framework* jMetal, desenvolvido na linguagem Java (Durillo and Nebro, 2011). É importante mencionar que o jMetal disponibiliza diversas metaheurísticas multiobjetivas documentadas na literatura e provê uma arquitetura que ajuda na adaptação, extensão e reuso para diferentes problemas de busca (Durillo *et al.*, 2011).

Em relação aos operadores genéticos da abordagem original de Lima e Elias (2017), a abordagem aqui proposta fez algumas adaptações para melhor se adequar a problemas multiobjetivos, a saber: *i)* ***população inicial***: é criada aleatoriamente, mas a probabilidade de escolher um determinado bug b_i é inversamente proporcional ao custo acumulado, definido pelo somatório dos custos do próprio bug b_i e dos demais bugs b_j que compõem suas dependências diretas e indiretas em $BX_{B\times B}$. Caso o custo acumulado do bug b_i seja maior que o orçamento disponível, o bug não é escolhido. *ii)* ***seleção de pais***: para cada pai, duas soluções presentes na população são sorteadas aleatoriamente e é escolhida a solução dominante, de acordo com o conceito de dominância de Pareto. Caso haja empate, é escolhida aleatoriamente uma das duas soluções. *iii)* ***probabilidade de mutação***: é inversamente proporcional à quantidade de bugs que possuem custo acumulado menor ou igual ao orçamento em questão. Note que, caso o custo acumulado do bug b_i seja maior que o orçamento disponível, o bug não é escolhido para mutação.

Os experimentos foram realizados sobre um grupo de 20 bugs e outro grupo de 50 bugs para uma análise das melhores soluções recomendadas pela abordagem multiobjetiva. No caso de 20 bugs, foi possível executar o algoritmo exaustivo, que identificou as soluções não-dominadas na fronteira de Pareto, representando as soluções

ótimas para o problema. Já no caso de 50 bugs, não foi possível executar o algoritmo exaustivo devido ao tamanho do espaço de busca, que acarretaria um tempo inviável de execução, na ordem de milhares de anos. Então, alternativamente, executou-se o algoritmo multiobjetivo 10.000 vezes, extraindo como fronteira de Pareto de referência as soluções não-dominadas dentre todas as execuções.

Ao observar a fronteira de Pareto para 20 bugs e a fronteira de Pareto de referência para 50 bugs, percebeu-se que o número de soluções não-dominadas é muito pequeno em todos os orçamentos avaliados. Por exemplo, no caso de 20 bugs, uma única solução foi encontrada em seis orçamentos, enquanto duas soluções foram encontradas nos demais orçamentos. Já para o caso de 50 bugs, uma única solução foi encontrada em quatro orçamentos, entre duas e quatros soluções foram encontradas nos demais orçamentos, e, no melhor cenário, cinco soluções foram encontradas em um único orçamento.

Consequentemente, como as fronteiras possuem poucas soluções, não é adequado adotar o espalhamento como indicador de qualidade. Logo, escolheu-se o hipervolume normalizado HVR para avaliar a qualidade das soluções encontradas, permitindo observar o quão próximas as soluções recomendadas estão das fronteiras de Pareto real e de referência, nos casos de 20 e 50 bugs, respectivamente.

Na parametrização dos experimentos, no caso de 20 bugs, adotou-se uma população de 40 indivíduos e 80 gerações, avaliando 3.200 soluções candidatas. Já no caso de 50 bugs, adotou-se uma população de 100 indivíduos e 200 gerações, chegando a 20.000 soluções candidatas avaliadas. Note que o total de soluções candidatas avaliadas, em ambos os casos, é bem pequeno em relação ao espaço de busca de 2^{20} e 2^{50} soluções.

Pela natureza estocástica do algoritmo NSGA-II, realizou-se cada experimento 10 vezes para cada nível de orçamento, a fim de obter uma média do HVR e do tempo de execução. Obviamente, não houve necessidade de repetição do algoritmo exaustivo com 20 bugs para encontrar a fronteira de Pareto real, e, com 50 bugs, percebeu-se que as 10.000 execuções já eram suficientes para encontrar a fronteira de Pareto de referência.

No experimento com 20 bugs, como mostrado na Tabela 2, obteve-se HVR médio de 1,0 para todos os orçamentos, com exceção do orçamento de 20% que obteve 0,9846, indicando que a fronteira de Pareto foi encontrada em todos os casos, exceto no orçamento de 20%, que ficou bastante próxima. Em relação ao tempo de execução médio, a menor duração foi no orçamento de 50% com 0,1795 segundos, enquanto a maior duração foi no orçamento de 75% com 0,1992 segundos.

É importante destacar que, com 20 bugs, para encontrar a fronteira de Pareto real, o algoritmo exaustivo consumiu 16 minutos e 16 segundos de processamento no orçamento de 100%, representando a maior duração dentre os orçamentos. Ademais, a menor duração ocorreu no orçamento de 5% com 1,16 segundo.

No experimento com 50 bugs, como também mostrado na Tabela 2, obteve-se HVR médio de 1,0 em nove orçamentos, e entre 0,9999 e 0,9801 nos demais orçamentos, indicando que a fronteira de Pareto de referência foi encontrada na maioria dos casos, exceto em cinco orçamentos, que ficou bastante próxima. Observe que, dentre os 14 orçamentos avaliados, em apenas cinco orçamentos não foram obtidos HVR máximo igual a 1,0. Contudo, mesmo nestes cinco casos, os valores do HVR não se distanciam mais que 2%. Ademais, em relação ao tempo de execução médio, a menor e a maior duração foram de 5,7741 e 6,0102 segundos, nos orçamentos de 5 e 25%, respectivamente.

É importante destacar que, com 50 bugs, para encontrar a fronteira de Pareto de referência, o algoritmo NSGA-II consumiu 18 minutos e 5 segundos de processamento no orçamento de 95%, representando a menor duração dentre os orçamentos. Já a maior duração ocorreu no orçamento de 100% com 19 minutos e 58 segundos. Em geral, encontrar a fronteira de referência acaba consumindo cerca de 200 vezes mais tempo.

Tabela 2. Resultados dos experimentos de 20 e 50 bugs

Cenário com 20 bugs		
Orçamento	**HVR**	**Duração**
5%	1,0000	0,1860
10%	1,0000	0,1979
20%	0,9846	0,1945
25%	1,0000	0,1942
30%	1,0000	0,1826
40%	1,0000	0,1949
50%	1,0000	0,1795
60%	1,0000	0,1980
70%	1,0000	0,1873
75%	1,0000	0,1992
80%	1,0000	0,1964
90%	1,0000	0,1841
95%	1,0000	0,1905
100%	1,0000	0,1922

Cenário com 50 bugs		
Orçamento	**HVR**	**Duração**
5%	1,0000	5,7741
10%	1,0000	5,8081
20%	0,9888	5,7917
25%	0,9801	6,0102
30%	1,0000	5,9735
40%	0,9999	5,9070
50%	0,9804	5,8413
60%	1,0000	5,9644
70%	1,0000	5,8060
75%	0,9997	5,9851
80%	1,0000	6,0062
90%	1,0000	5,9631
95%	1,0000	5,8775
100%	1,0000	5,9094

Em conclusão, no caso de 20 bugs, as soluções não-dominadas recomentadas quase sempre representam as fronteiras de Pareto reais, sendo as execuções do algoritmo NSGA-II extremamente rápidas em relação ao algoritmo exaustivo. Nessa mesma linha, também segue o caso de 50 bugs, cujas soluções não-dominadas recomendas são muito próximas das fronteiras de Pareto de referência, sendo as execuções do algoritmo NSGA-II também bastante rápidas em relação às 10.000 execuções requeridas para encontrar a respectivas fronteiras. Portanto, a abordagem multiobjetiva proposta é efetiva e eficiente para recomendar excelentes soluções para o Problema de Seleção de Bugs.

6. Conclusão

O presente trabalho apresenta uma abordagem multiobjetiva para o Problema de Seleção de Bugs, com resultados que muito se aproximam do ótimo, quando conhecidos, recomendando quase sempre soluções não-dominadas das fronteiras de Pareto reais e de referência, obtidas em execuções comparativas muito demoradas do algoritmo exaustivo ou inúmeras repetições do algoritmo NSGA-II. Portanto, além da qualidade das soluções recomendadas, a abordagem proposta também apresenta celeridade no tempo de execução, quando comparada com as execuções comparativas. Em conclusão, a abordagem aqui adaptada e proposta é bastante efetiva e eficiente em solucionar o problema na perspectiva multiobjetiva, principalmente se for considerada a fração diminuta de soluções candidatas que são avaliadas em relação espaço de busca total.

Ademais, assim como a proposta original de Lima e Elias (2017), a abordagem aqui proposta traz também como vantagem a relevância dada às dependências entre bugs definidas pela arquitetura de software, pois um conjunto mais coeso de bugs a serem

resolvidos auxilia a equipe de desenvolvimento (Almhana, 2020), uma vez que a arquitetura do sistema é essencial durante o desenvolvimento e manutenção do software.

Por fim, ao introduzir uma perspectiva multiobjetiva ao problema BSP, as soluções não-dominadas recomendadas incrementam o número de escolhas e nuances para a tomada de decisão, proporcionando um compromisso adequado entre os níveis de prioridade e severidade, sem necessariamente degradar nenhum deles.

Entretanto, como limitações, primeiro, a adoção de apenas um único algoritmo multiobjetivo, no caso o NSGA-II, pode criar vieses nos resultados para as partes efetivas do algoritmo e não analisar e perceber suas deficiências. Segundo, por considerar um único projeto de software, no caso o sistema SIGAA, as calibrações adotadas nos experimentos, obtidas de forma empírica, podem certamente não ser adequadas em outros projetos mais complexos. E, terceiro, a falta de avaliação em relação à competividade humana, onde procura-se comparar os resultados obtidos de forma automática pela metaheurística e os resultados obtidos manualmente por gerentes de projeto.

Logo, como trabalhos futuros, é interessante também avaliar a abordagem proposta com outras metaheurísticas evolucionárias multiobjetivas, reforçando assim a consistência da modelagem matemática, e, possivelmente, até permitindo encontrar mais soluções não-dominadas nas fronteiras de Pareto de referência, unindo as soluções encontradas nos diferentes algoritmos. Como outra melhoria, seria interessante aplicar a abordagem proposta em outros projetos de software, tornando possível melhorar a confiança e até definir regras de calibração mais generalizadas. Por fim, também seria importante realizar um estudo de competividade humana, evidenciando que a adoção da abordagem proposta pode reduzir o tempo e melhorar a qualidade das soluções na tarefa de seleção de bugs para correção.

Referências

Almhana, R., Mkaouer, W., Kessentini, M., and Ouni, A. (2016). Recommending relevant classes for bug reports using multi-objective search. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, pp. 286-295.

Almhana, R. and Kessentini, M. (2021). Considering dependencies between bug reports to improve bugs triage. *Automated Software Engineering, 28*, 1-26.

Anvik, J., Hiew, L., and Murphy, G. C. (2006). Who should fix this bug?. In *Proceedings of the 28th International Conference on Software Engineering*, pp. 361-370.

Chopard, B., and Tomassini, M. (2018). *An introduction to metaheuristics for optimization,* pp. 191-203. Cham, Switzerland: Springer.

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. A. M. T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation, 6*(2), 182-197.

Dreyton, D., Araújo, A. A., Dantas, A., Saraiva, R., and Souza, J. (2016). A multi-objective approach to prioritize and recommend bugs in open source repositories. In *Proceedings of the 8th International Symposium on Search Based Software Engineering*, pp. 143-158. Springer International Publishing.

Durillo, J. J., and Nebro, A. J. (2011). jMetal: A Java framework for multi-objective optimization. *Advances in Engineering Software, 42*(10), 760-771.

Durillo, J. J., Zhang, Y., Alba, E., Harman, M., and Nebro, A. J. (2011). A study of the bi-objective next release problem. *Empirical Software Engineering, 16*, 29-60.

Eclipse Foundation (2024). Eclipse IDE. Last access on April 2024 at: https://eclipseide.org.

Harman, M., Mansouri, S. A., and Zhang, Y. (2012). Search-based software engineering: trends, techniques and applications. ACM Computing Surveys, 45(1), 1-61.

Lima, G., and Elias, G. (2017). An automated approach for selecting bugs in component-based software projects. In *Proceedings of the International Conference on Software Engineering Research and Practice*, pp. 38-44.

Lima, J. A. (2019). *Uma abordagem para seleção e alocação de pessoas baseada em perfis técnicos e de personalidades para projetos de software*. Dissertação de Mestrado, Universidade Federal da Paraíba. Disponível em: https://repositorio.ufpb.br/jspui/handle/123456789/19617?locale=pt_BR.

Ma, H., Zhang, Y., Sun, S., Liu, T., and Shan, Y. (2023). A comprehensive survey on NSGA-II for multi-objective optimization and applications. *Artificial Intelligence Review, 56*(12), 15217-15270.

Mozilla (2024). Firefox Browsers. Last access on April 2024 at: https://www.mozilla.org/en-US/firefox.

Srinivas, N., and Deb, K. (1994). Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary computation, 2*(3), 221-248.

Warfield, J. N. (1973). Binary matrices in system modeling. *IEEE Transactions on Systems, Management, and Cybernetics*, 3(5), 441-449.

ZBC (2024). Bugzilla. Last access on April 2024 at: https://www.bugzilla.org.