13° congresso de pesquisa, ensino e extensão conpeex

Ciência alimentando o Brasil

ANAIS DO XIII CONPEEX

Congresso de Pesquisa, Ensino e Extensão
Universidade Federal de Goiás

De 17 a 19 de outubro de 2016

PROGRAMA INSTITUCIONAL DE
BOLSA DE INICIAÇÃO CIENTÍFICA

PIBIC

Apoio: FUNAPE FAPEG SED GOIÁS CAIXA BRASIL

Realização: UFG

Aluno	Trabalho
EDUARDO FARIA DE SOUZA	Um módulo para visualizar a evolução genotípica na ferramenta GenProg
EDUARDO HENRIQUE SANTOS	AVALIACAO EM MUSICA
ELIZA MURIELE TEIXEIRA DA SILVA	CARACTERIZAÇÃO DOS SOLOS DESFLORESTADOS NAS ÁREAS DE PRESERVAÇÃO PEMANENTE (APP) DO RIO CLARO.
ELMAR SEVERINO RIBEIRO JUNIOR	IMAGINÁRIO SOCIAL E REPRESENTAÇÕES CULTURAIS NO LIVRO ESCOLAR PRIMÁRIO SEGUNDO LIVRO DE LEITURA PARA A INFÂNCIA: NA ESCOLA E NO LAR, DE THOMAZ GALHARDO (59ª edição, 1938) ¿ primeiras noções de cultura e cidadania.
ERICK LUCIANO FERREIRA	JORNALISMO, LITERATURA E HISTÓRIA NA EUROPA FASCISTA NA OBRA ¿AFIRMA PEREIRA¿ DE ANTONIO TABUCCHI
ERIKA VILELA VALENTE	DIAGNÓSTICO AMBIENTAL DA MICROBACIA DO CÓRREGO ARROZAL NO MUNICÍPIO DE TRINDADE-GO.
EUSLAN DE ALMEIDA JUNIOR	AQUISIÇÃO DE FERRO MEDIADA POR SIDERÓFOROS EM CLADOPHIALOPHORA CARRIONII
FERNANDA AZEVEDO SILVA	A Transfiguração em arte: Uma introdução à Arthur Danto
FERNANDA MARIA OLIVEIRA AGUIAR	ANÁLISE DE CUSTO-EFETIVIDADE DE DUAS MODALIDADES DE TRATAMENTO PARA DESDENTADOS TOTAIS: PRÓTESE TOTAL CONVENCIONAL E OVERDENTURE MANDIBULAR RETIDA POR IMPLANTE UNITÁRIO
FERNANDA NARUMI MIYAGI YOSHIHARA	Otimização da produção de uma celulase recombinante (CBH 1.2) pelos transformantes de Pichia pastoris

UM MÓDULO PARA VISUALIZAR A EVOLUÇÃO GENOTÍPICA NA FERRAMENTA GENPROG

SOUZA, Eduardo Faria de[1]; **CAMILO-JÚNIOR**, Celso Gonçalves[2]; **LE GOUES**, Claire[3]; **RODRIGUES**, Cássio Leonardo[4];

Palavras-chave: Visualização, Algoritmo genético, Reparo automatizado de software

1. Justificativa

A tarefa de manutenção de software, especialmente a parte de consertar defeitos, é uma das mais onerosas da indústria de software. Seja pelo custo ou pelo impacto negativo na marca de uma empresa, defeitos em software são problemas de impacto global. Sendo assim, muito esforços são feitos para ajudar as desenvolvedoras e empresas. Dentro dos trabalhos da área de Reparo Automatizado de Software (WEIMER et. al., 2010) a GenProg (LE GOUES et. al., 2012) é um dos métodos mais citados. Ela é baseada no modelo de Computação Evolucionária e tem como objetivo a geração de reparos sem intervenção humana, seja durante o conserto ou na instrumentação prévia. Entretanto, apesar do progresso, o comportamento da evolução genética da GenProg é desconhecido.

2. Objetivos

Este trabalho propõe um módulo, chamado Painel VoR, para monitorar a evolução genotípica e, em consequência, a análise da busca em relação aos espaços de busca usados pela GenProg. Este painel consiste em um módulo integrado a GenProg destinado à extração de dados e um painel para a visualização dos dados obtidos. Esta ferramenta permite comparar os efeitos das mudanças feitas nos genótipos da GenProg e rastrear a evolução do algoritmo genético geração a geração. Alguns dos benefícios de usá-la são, por exemplo, ser possível observar a cobertura dos diferentes espaços de busca pelo algoritmo, destacar os genes privilegiados, e ajudar no refinamento dos parâmetros. Vale ressaltar que nenhuma dessas informações são facilmente visualizadas sem a ajuda de uma ferramenta especializada como a aqui apresentad. Portanto, a maior contribuição deste artigo é uma ferramenta para ajudar as desenvolvedoras da GenProg a melhorarem seu algoritmo genético para endereçar diferentes problemas.

[1] Bolsista. Instituto de Informática – UFG. eduardosouza@inf.ufg.br
[2] Professor associado. Instituto de Informática – UFG. celso@inf.ufg.br
[3] Assistant Professor. School of Computer Science – Carnegie Mellon University. clegoues@cs.cmu.edu
[4] Orientador. Professor associado. Instituto de Informática – UFG. cassio@inf.ufg.br

3. Metodologia

Esta pesquisa se baseia na metodologia de construção. Seguindo os passos de revisão bibliográfica, estudo da linguagem de programação OCaml, estudo da ferramenta GenProg, desenvolvimento do painel, e, por fim, as validações e testes da ferramenta.

4. Resultados e discussão

Observar o comportamento da GenProg em detalhes a partir de sua saída padrão pode não ser tarefa trivial. Assim como comparar duas execuções distintas usando o mesmo recurso pode não levar a conclusões claras. Essa dificuldade é decorrente de como a apresentação destes dados é feita. Portanto, ao utilizar o Painel VoR, consegue-se atingir esse nível de detalhamento na observação do comportamento interno do algoritmo genético e fazer tomadas de decisões com melhor embasamento.

5. Conclusão

A GenProg é uma ferramenta de Reparo Automatizado de Software baseada em programação genética. Contudo, por ter uma saída padrão simples, é difícil de visualizar como seu algoritmo genético funciona internamente. Então, este trabalho apresentou uma ferramenta capaz de coletar dados da execução da GenProg e então mostrá-los como gráficos. Estes gráficos ajudam a entender melhor o comportamento interno da ferramenta, verificando seu está como o esperando ou como pode ser melhorada.

6. Referências

Weimer, Westley, et al. "Automatic program repair with evolutionary computation." *Communications of the ACM* 53.5 (2010): 109-116.

Le Goues, Claire, et al. "Genprog: A generic method for automatic software repair." *IEEE Transactions on Software Engineering* 38.1 (2012): 54-72.

Um módulo para visualizar a evolução genotípica na ferramenta GenProg

Eduardo F. de Souza[1]*, Cassio Rodrigues[1]†, Claire Le Goues[2], Celso G. Camilo Junior[1]

[1]Instituto de Informática – Universidade Federal de Goiás (UFG)
Goiânia – GO – Brazil

[2]School of Computer Science – Carnegie Mellon University (CMU)
Pittsburgh – PA – USA

{eduardosouza,cassio,celso}@inf.ufg.br, clegoues@cs.cmu.edu

Resumo. *A tarefa de manutenção de softwares, especialmente a parte de consertar defeitos, é uma das mais onerosas da indústria de software. Seja pelo custo ou pelo impacto negativo na marca, defeitos em software são problemas de impacto global. Sendo assim, muito esforços são feitos para ajudar as desenvolvedoras e empresas. Dentro dos trabalhos da área de Reparo Automatizado de Software a GenProg é um dos métodos mais citados. Ela é baseada no modelo de Computação Evolucionária e tem como objetivo a geração de reparos com intervenção humana, seja durante o conserto ou na instrumentação prévia. Entretanto, apesar do progresso, o comportamento da evolução genética da GenProg é desconhecido. Então, este trabalho propõe um módulo, chamado Painel VoR, para monitorar a evolução genotípica e, em consequência, a análise da busca em relação aos espaços de busca usados pela GenProg. Para avaliar a proposta, um caso de uso é avaliado baseado em gráficos e outras informações do painel. Como resultado alguns benefícios são vistos quando usados o módulo, especialmente para aperfeiçoamento de parâmetros para cada problema.*

1. Introduction

Engenharia de Software, como um todo, pode ser considerada "a aplicação de uma abordagem ao desenvolvimento e manutenção de softwares que seja sistemática, disciplinada, e quantificável" [Bourque et al. 2014]. Aplicar boas praticas de Engenharia de Software ao desenvolvimento ajuda a mitigar os problemas naturais da prática. Entretanto, manutenção de sistema é tipicamente mais onerosa do que o desenvolvimento em si, custando até 70% do ciclo de vida de um software [Pressman 2005]. Como resultado, pesquisadoras buscam técnicas para reduzir os custos com manutenção.

Dependendo do tamanho do projeto, algumas tarefas tornam-se quase impossíveis para serem realizadas manualmente, motivando o desenvolvimento de técnicas automatizadas. Uma área da Engenharia de Software que busca automatizar essas ações é conhecida como Engenharia de Software Baseada em Busca. Esta área trata os problemas da Engenharia de Software como problemas de busca, que então podem ser modelados como problemas de maximização ou minimização. Isso para buscar soluções

*Bolsista
†Orientador

ótimas (ou sub-ótimas) através da aplicação de técnicas de busca ou otimização. Estas soluções estão situadas em um espaço de busca cheio de possíveis soluções (ou soluções candidatas). A busca, então, será guiada por uma função objetivo que distinguirá as boas soluções das ruins [Harman and Jones 2001]. As técnicas dessa área tem sido aplicadas para problemas de seleção de requisitos [Zhang 2010], otimização de software automatizada [Langdon and Harman 2013], e reparo automatizado de software [Weimer et al. 2009].

Falhas em software são de considerável impacto social e econômico. O Instuto Nacional de Padrões e Tecnologia (NIST) do Estados Unidos estima que a economia estadunidense perde US$60 bilhões a cada ano desenvolvendo e aplicando correções para as falhas dos softwares [Zhivich and Cunningham 2009]. Quando falhas atingem sistemas de missão crítica elas podem levar a grandes desastres. As três falhas mais conhecidas dessa categoria foram o apagão que aconteceu em 2003 nos Estados Unidos, a falha na máquina de radiação Thrac-25, e a explosão do foguete Ariane 5. O apagão ocorreu na região nordeste dos Estados Unidos devido à um laço de repetição infinita que estava presente no sistema de gerenciamento de alarmes, provocando um dano estimado de US$10 bilhões [Zhivich and Cunningham 2009]. Nos anos 1980, uma falha aritmética na Therac-25, uma máquina de radioterapia usada para tratar câncer, foi responsável pela morte de seis pacientes devido a uma exposição intensão à radiação [Leveson and Turner 1993]. O foguete Ariane 5, desenvolvido pela Agência Espacial Europeia a um custo de US$1 bilhão, foi destruído 40 segundos após sua decolagem em 4 de junho 1996. As investigadoras descobriram que o desastre foi resultado de erros na especificação e no projeto do software que era responsável por calcular a altitude do foguete [Lions et al. 1996].

Apesar de todas essas falhas terem impacto em sistemas de missão crítica, elas também são onerosas para sistemas que não são dessa categoria. O ano de 2014 foi marcado por duas vulnerabilidades críticas, o *Heat Bleed* (CVE-2014-0160) e o *Shellshock* (CVE-2014-6271). Elas foram falhas em softwares que estão no núcleo do funcionamento da maioria dos computadores, a ferramenta que provê segurança SSL/TLS chamada OpenSSL, e o GNU Bash, respectivamente. Esses dois softwares são utilizados desde os servidores de grandes empresas até os celulares que carregamos.

Para mitigar os prejuízos causados pelas falhas, um grupo de pesquisadoras de diferentes universidades propuseram uma ferramenta de Engenharia de Software Baseada em Busca para o reparo automatizado de software chamada GenProg [Le Goues et al. 2012b]. Esta ferramenta é baseada na meta-heurística da Programação Genética [Banzhaf et al. 1998], e tem por objetivo corrigir falhas de softwares de forma eficiente e com a menor intervenção humana de forma que estes não tenham suas funcionalidades existentes quebradas. Com isso é possível reduzir os custos do desenvolvimento de softwares e também o tempo necessário para reparar uma falha. A qualidade final é aumentada, que leva a empresas e consumidores a se beneficiarem ao usarem este processo automatizado. O código fonte da GenProg está disponível em `http://genprog.cs.virginia.edu/`.

A GenProg modela seus *genótipos* como sequências de operações realizadas no código (também chamadas de *patches*), e seus *fenótipos* como as possíveis soluções. Estes *patches* são sequências de operações de edição a nível de instruções de código. As variantes são esses *patches* aplicados ao código original, resultando em um código fonte mo-

dificado [Le Goues et al. 2012a]. A busca é realizada através de três espaços que, quando combinados, resultam em um espaço de busca final. A função de aptidão padrão é dada pela soma da quantidade de casos de testes aplicados à variante que retornaram os resultados esperados. O modelo final desta arquitetura é complexo, portanto, fazer um detalhamento do comportamento desse algoritmo para entendê-lo e melhorá-lo pode ser difícil sem a ajuda de uma ferramenta especializada. Apesar de a literatura ter algumas ferramentas para a visualização de algoritmos genéticos ([Barlow et al. 2002], [Pohlheim 1999]), nenhuma delas é direcionada para ser aplicada com a GenProg e prover informações úteis a partir das características específicas da ferramenta.

Dado o contexto apresentado, este trabalho se destina a responder às seguintes perguntas de pesquisa:

- **PP1:** Dadas as características da GenProg, é possível construir uma ferramenta que permita a visualização de funcionamento interno?
- **PP2:** Como o algoritmo de busca está se comportando dentro dos espaços de busca?
- **PP3:** Como essa ferramenta pode ajudar às suas desenvolvedoras a encontrarem pontos que possivelmente precisam ser melhorados?

Este trabalho propões uma ferramenta chamada Painel VoR, que consistem em um módulo integrado a GenProg destinado à extração de dados e um painel para a visualização dos dados obtidos. Esta ferramenta permite comparar os efeitos das mudanças feitas nos genótipos da GenProg, e rastrear a evolução do algoritmo genético geração a geração. Alguns dos benefícios de usá-la são, por exemplo, ser possível observar a cobertura dos diferentes espaços de busca pelo algoritmo, destacar os genes privilegiados, e ajudar no refinamentos dos parâmetros. Vale ressaltar que nenhuma dessas informações são facilmente visualizadas sem a ajuda de uma ferramenta especialidade como a apresentada neste artigo. Portanto, a maior contribuição deste artigo é uma ferramenta para ajudar as desenvolvedoras da GenProg a melhorarem seu algoritmo genético para endereçar diferentes problemas.

O restante deste artigo é organizado da seguinte forma: Seção 3 mostra o que é Reparo Automatizado de Software e sua importância na Engenharia de Software; Seção 4 descreve como a GenProg funciona, suas conquistas mais notáveis até agora, e explica o problema da visualização de seu algoritmo genético; Seção **??** descreve a principal contribuição do artigo, que é o Painel VoR; discussões a respeito da ferramenta e respostas às perguntas de pesquisa estão na Seção 6; seguido pela Seção 7 que mostra os trabalho correlatos; e finalmente, a conclusão se situa na Seção 8.

2. Reparo Automatizado de Software

O número de falhas em um software pode sobrepor a capacidade de uma desenvolvedora de corrigí-las. Além disso, algumas falhas podem ficar até 15 anos sem serem corrigidas [Bugzilla 2015], ou até serem corrigidas de forma errônea. Um estudo analisou correções enviadas para corrigirem falhas em sistemas operacionais de código abertos e descobriu que de 14.8% a 24.4% dessas correções estão incorretas. Outra descoberta foi que 27% dessas desenvolvedoras que eviaram as correções não tinham experiência com os códigos que estavam trabalhando [Yin et al. 2011]. Algumas delas, ao tentarem corrigir

as falhas, acabaram deteriorando o software ao introduzirem novas falhas. Estes problemas acabam afetando a confiabilidade do sistema. Precisamos, então, de um método automatizado para corrigir essas falhas que não necessite de pessoas para realizá-lo. Isso poderia previr vários dos problemas apresentados, ao passo que aumentaria a qualidade e confiabilidade do software resultante deste processo, além de reduzir os custos com manutenção.

Reparo Automatizado de Software (RAS) é uma área de pesquisa que busca corrigir falhas em software de maneira automatizada e com a menor intervenção possível. Ao caminhar para esses objetivos é possível reduzir custos com manutenção e aumentar a resiliência dos softwares à falhas e situações não esperadas [DeMarco et al. 2014]. A correção é, geralmente, codificada como uma mudança incremental no código fonte (também chamada de "patch"ou "diff") [Martinez and Monperrus 2013]. Tais *patches* são aplicados ao código em níveis de intruções de código [DeMarco et al. 2014, Le Goues et al. 2012b], ou em níveis de estados [Perkins et al. 2009].

Várias técnicas podem ser usadas para buscar por uma correção, mas todas precisam de um oráculo de reparo para identificar tanto a falha a ser reparada quanto o comportamento do software que deve ser preservado. A GenProg usa a computação evolucionária como sua técnica de busca por correções [Le Goues et al. 2012b], enquanto outra ferramenta conhecida da área, chamada Nopol, se baseia em solucionadores baseados em restrições [DeMarco et al. 2014]. O oráculo clássico é o conjunto de casos de teste, onde os casos de teste negativos agem como o oráculo de falha, e os positivos agem como um oráculo de regressão. Um *patch* é considerado uma correção quando todos os casos de testes, positivos e negativos, retornam os valores esperados.

3. Reparo Automatizado de Software

O número de falhas em um software pode sobrepor a capacidade de uma desenvolvedora de corrigí-las. Além disso, algumas falhas podem ficar até 15 anos sem serem corrigidas [Bugzilla 2015], ou até serem corrigidas de forma errônea. Um estudo analisou correções enviadas para corrigirem falhas em sistemas operacionais de código abertos e descobriu que de 14.8% a 24.4% dessas correções estão incorretas. Outra descoberta foi que 27% dessas desenvolvedoras que eviaram as correções não tinham experiência com os códigos que estavam trabalhando [Yin et al. 2011]. Algumas delas, ao tentarem corrigir as falhas, acabaram deteriorando o software ao introduzirem novas falhas. Estes problemas acabam afetando a confiabilidade do sistema. Precisamos, então, de um método automatizado para corrigir essas falhas que não necessite de pessoas para realizá-lo. Isso poderia previr vários dos problemas apresentados, ao passo que aumentaria a qualidade e confiabilidade do software resultante deste processo, além de reduzir os custos com manutenção.

Reparo Automatizado de Software (RAS) é uma área de pesquisa que busca corrigir falhas em software de maneira automatizada e com a menor intervenção possível. Ao caminhar para esses objetivos é possível reduzir custos com manutenção e aumentar a resiliência dos softwares à falhas e situações não esperadas [DeMarco et al. 2014]. A correção é, geralmente, codificada como uma mudança incremental no código fonte (também chamada de "patch"ou "diff") [Martinez and Monperrus 2013]. Tais *patches* são aplicados ao código em níveis de intruções de código [DeMarco et al. 2014,

Le Goues et al. 2012b], ou em níveis de estados [Perkins et al. 2009].

Várias técnicas podem ser usadas para buscar por uma correção, mas todas precisam de um oráculo de reparo para identificar tanto a falha a ser reparada quanto o comportamento do software que deve ser preservado. A GenProg usa a computação evolucionária como sua técnica de busca por correções [Le Goues et al. 2012b], enquanto outra ferramenta conhecida da área, chamada Nopol, se baseia em solucionadores baseados em restrições [DeMarco et al. 2014]. O oráculo clássico é o conjunto de casos de teste, onde os casos de teste negativos agem como o oráculo de falha, e os positivos agem como um oráculo de regressão. Um *patch* é considerado uma correção quando todos os casos de testes, positivos e negativos, retornam os valores esperados.

4. GenProg

GenProg é uma ferramenta de Reparo Automatizado de Software. É genérica por conseguir reparar diferentes tipos de defeitos, e é automatizada por operar sem intervenção humana. Ela pode ser definida como "uma técnica que usa casos de testes existentes para automaticamente gerar reparos para falhas presentes em softwares legados" [Le Goues et al. 2012b]. A ferramenta recebe como entrada um programa com defeito e um conjunto de casos de testes, divididos em positivos e negativos. Usando programação genética, ela gera versões modificadas a partir do código original, que são chamadas variantes. A função de aptidão aplica os casos de testes em cada variante para obter suas avaliações. Como resultado, a variante que mantém o comportamento especificado e faz todos as falhas desaparecerem é considerada uma correção [Le Goues et al. 2012b, Weimer et al. 2010]. Manter o comportamento especificado significa passar em todos os casos de testes positivos, enquanto desaparecer com as falhas significa passar em todos os casos de testes negativos.

Cada variante é representada como um *patch*, que é uma sequência de operações de edição no código original a nível de instruções. O operador de mutação é responsável por introduzir edições aleatórias (inserção, troca, e remoção) a uma variante. A operação de inserção seleciona uma instrução de um conjunto previamente gerado a partir das próprias instruções do código original e a coloca após uma outra instrução do código da variante. A operação de troca seleciona duas instruções do código da variante a as troca de lugar. Por fim, a remoção seleciona uma instrução do código da variante e a remove [Le Goues et al. 2012a].

Estas operações são, portanto, construídas pela composição de elementos provenientes de três espaços de busca: o espaço de falhas, o espaço de reparos, e o espaço das operações. O espaço de falhas consiste das instruções que possivelmente estão causado os erros. O espaço de reparos consiste em um conjunto de isntruções provenientes do código original. O espaço das operações é composto pelas operações de inserção, troca, e remoção.

Um dos artigos da GenProg se destaca. Ele reporta uma acurácia de aproximadamente 50% ao tentar reparar diferentes tipos de defeitos em vários softwares de código aberto. Em média, cada um desses reparos custou US$8 cada [Le Goues et al. 2012a]. Apesar de a ferramenta conseguir reparar alguns bugs em uma base de código genérica, ainda existe muitos problemas que ela não resolve. O trabalho [Le Goues et al. 2013] mostra que uma pesquisa mais extensa juntamente com a continuação do desenvolvimento

da ferramenta ainda são necessários. Por exemplo, o método atual de calcular a aptidão de cada variante talvez não seja o melhor em alguns casos. Talvez uma uma avaliação mais detalhada de cada variante poderia guiara a busca em direção a uma correção com mais acurácia e rapidez.

A saída padrão da GenProg, como mostrada na Figure 1, não está em um formato que seja fácil extrair informações sobre a execução, e também não tem todas as informações do algoritmo genético presente. Com isso, o desafio principal para gerar boas visualizações, neste caso, e coletar a maior quantidade de dados possíveis sobre o algoritmo genético.

```
search: genetic algorithm begins (|original| = 0.00631428 MB)
search: initial population (sizeof one variant = 0.00624466 MB)
          0 s(1,13)
          5 d(1)
          5 d(3)
          5 d(13)
          5 original
search: generation 1 (sizeof one variant = 0.00705147 MB)
          0 d(1) d(10)
          5 d(13) a(1,9)
          5 d(1) d(1)
          5 d(3) a(7,14)
search: generation 2 (sizeof one variant = 0.00707245 MB)
          5 d(3) a(7,14) d(1)
          5 d(3) a(7,14) d(1)
          5 d(3) a(7,14) d(1)
          5 d(3) a(7,14) a(1,3)
          1 d(1) d(1) d(11)
          1 d(3) a(7,14) s(11,8)
```

Figura 1. Excerto da saída padrão da GenProg

Resolver este problema permite observar o impacto das taxas probabilísticas dos operadores, verificar a utilização dos espaços de busca, e qual o impacto da atual função de aptidão através das gerações.

5. VoR Panel

O Painel VoR é uma ferramenta para visualizar o comportamento do algoritmo genético da GenProg. Ele é composto por um módulo integrado a GenProg para extração de dados, e de uma ferramenta de visualização.

O módulo de extração de dados faz a coleta primeiro uma coleta de informações estáticas, que são a semente aleatória utilizada e as probabilidades dos operadores. Logo após essa coleta inicial, o módulo passa a coletar informações de cada geração como quais instruções foram usadas do conjunto de possíveis reparos, quais instruções foram alteradas no código da variante, quais operações foram utilizadas, e qual a aptidão da variante. Boa parte dessas informações são coletadas do *patch* da variante. Os últimos dados a serem coletados são a quantidade de gerações que foram executadas e se a execução resultou em um reparo ou não.

Esses dados são passados para a ferramenta de visualização que os processa e os mostra nos seguintes gráficos: (1) um histograma de avaliação geração-a-geração

da utilização da origem de cada instrução; (2) o mesmo histograma para as operações; (3) a evolução das aptidões através das gerações; (4) e a diversidade da utilização das instruções.

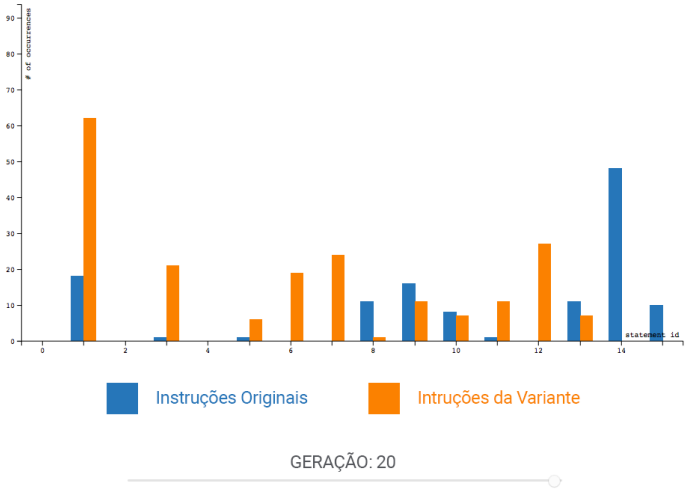

Figura 2. Distribuição das instruções.

1. A utilização de cada instrução pode ser visualizada na Figura 2. Este gráfico é separado por instruções que vieram do código original e as provenientes da variante. Cada elemento do eixo horizontal corresponde a uma instrução. Como essas instruções são enumeradas, então a coluna 1 corresponde à primeira instrução do código e assim por diante. O eixo vertical contabiliza o número de ocorrências da instrução em uma determinada geração. Logo abaixo do gráfico existe uma barra com um botão que, ao ser movimentado, mostra informações de diferentes gerações, sequencialmente.

2. A utilização das operações está presenta na Figura 3, que é um gráfico similar ao da Figura 2. O eixo horizontal separa as três operações e o eixo vertical contabiliza suas respectivas ocorrências em cada geração. Seguindo o padrão, a barra inferior controla qual geração terá suas informações apresentadas.

3. A evolução das aptidões das variantes é mostrado na Figura 4. Neste gráfico o eixo horizontal corresponde a cada geração e o eixo vertical aos valores de aptidão. Nele é apresentado as informações da aptidão da melhor variante de cada geração e a média de todas as variantes (incluindo a melhor).

4. A diversidade, ou seja, a variedade de instruções únicas utilizadas em cada geração, é mostrada na Figura 5. Nesta apresentação é mostrada quantas instruções únicas foram utilizadas em cada geração e qual a fonte delas, se vieram do código original ou se vieram da própria variante. O eixo horizontal denota as gerações, enquanto o eixo vertical apresenta a quantidade de instruções únicas.

6. Discussão e Resultados

Observar o comportamento da GenProg em detalhes a partir de sua saída padrão (Figura 1) pode não ser tarefa trivial. Assim como comparar duas execuções distintas usando o mesmo recurso pode não levar a conclusões claras. Essa dificuldade é decorrente de

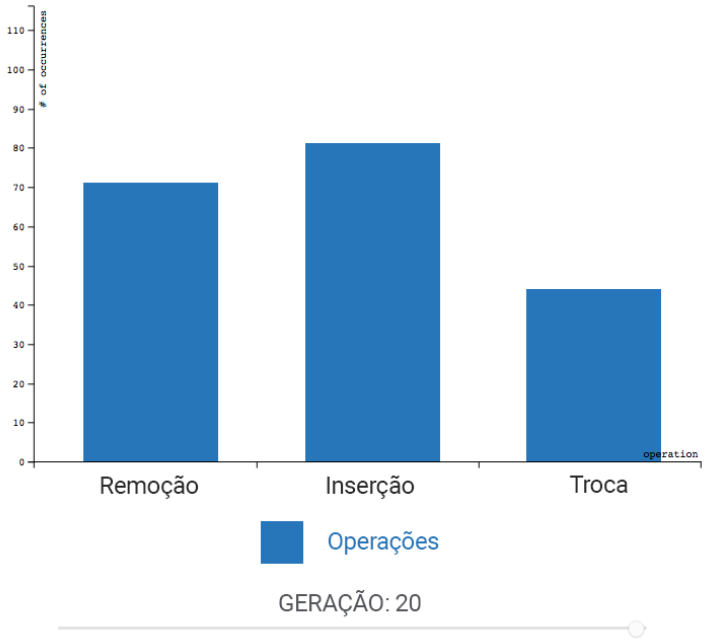

Figura 3. Distribuição das operações.

como a apresentação destes dados é feita. Portanto, ao utilizar o Painel VoR, consegue-se atingir esse nível de detalhamento na observação do comportamento interno do algoritmo genético.

É possível fazer um módulo integrado à GenProg que extraia dados internos de seu funcionamento e apresentá-lo de forma estruturada como gráficos. Portanto, tem-se a primeira **Pergunta de Pesquisa 1** respondida.

As informações obtidas nos gráficos dos histogramas das instruções e histogramas das operações mostram como o algoritmo está percorrendo os espaços de busca e quais são suas priorizações, é possível também, pelo gráfico da diversidade, observar o comportamento da convergência. Com essas informações conseguimos responder a **Pergunta de Pesquisa 2**.

Ao observar o gráfico das aptidões (Figura 4) nota-se que existem *plateaus*. Este comportamento é decorrente da maioria das variantes em uma geração apresentarem os mesmo valores de aptidão. Como essas variantes são distintas entre si, elas deveriam, então, apresentar aptidões diferentes. Fica evidente que a função de aptidão precisa ser melhorada, considerando avaliações mais granulares para evitar que duas ou mais variantes que sejam distintas tenham a mesma aptidão. Esta é uma observação que seria impraticável usando a saída padrão da GenProg em casos com centenas de variantes e de gerações. A **Pergunta de Pesquisa 3** fica então respondida por ser possível extrair hipóteses de melhoras para os algoritmos da GenProg a partir da observação dos gráficos.

7. Trabalhos correlatos

A literatura não provê ferramentas para visualizar o comportamento de algoritmos evolucionários no contexto de Reparo Automatizado de Software. Contudo, existem

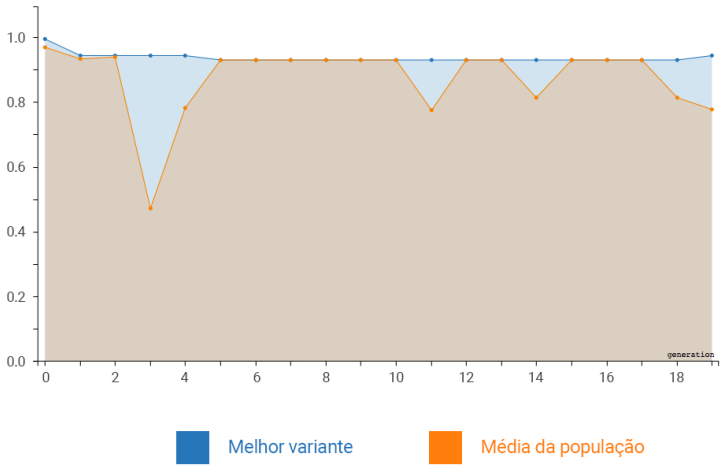

Figura 4. Evolução das aptidões.

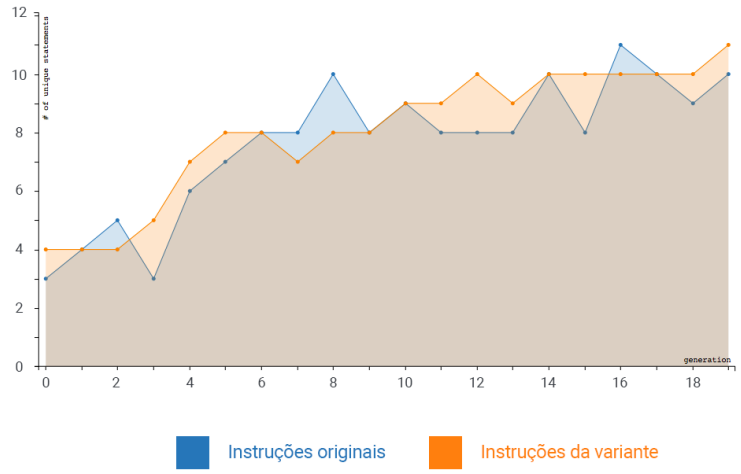

Figura 5. Evolução da diversidade (convergência).

trabalhos feitos ([Barlow et al. 2002], [Eisenmann et al. 2013], [Liao and Sun 2001], [Pohlheim 1999]) a fim de ajudar a visualizar o funcionamento interno dos algoritmos genéticos. Eles variam entre técnicas de comparação visual até ferramentas que ajudam na implementação dessas visualizações. Apesar de alguns deles tentarem apresentar formas abrangentes de visualizar os dados, nenhum é capaz de mostrar histogramas das variantes, evolução da aptidão, ou diversidade.

O Painel VoR consegue apresentar informações para vários espaços de busca e dispões os dados de cada variante em histogramas separados por gerações. Mas, apesar de trabalho feito em [Pohlheim 1999] ter apresentado "um conjunto padrão de técnicas para diferentes tipos de dados"e ter agregado-as no software GEATbx [Pohlheim 2015], ele não provê uma forma de apresentar as informações que o Painel VoR apresenta. O GEATbx também não consegue lidar com vários espaços de busca em suas visualizações. O mesmo ocorre com a [Barlow et al. 2002], que mostrou um conjunto de visualizações

usando uma ferramenta proprietária especializada em apresentação de grandes conjuntos de dados, mas, novamente, ela não considera o caso de vários espaços de busca.

8. Conclusão

A GenProg é uma ferramenta de Reparo Automatizado de Software baseada em programação genética. Contudo, por ter uma saída padrão simples é difícil de visualizar como seu algoritmo genético funciona internamente. Então, este artigo apresentou uma ferramenta capaz de coletar dados da execução da GenProg e então mostrá-los como gráficos. Estes gráficos ajudam a entender melhor o comportamento interno da ferramenta, verificando seu está como o esperando ou como pode ser melhorada.

Referências

Banzhaf, W., Nordin, P., Keller, R. E., and Francone, F. D. (1998). *Genetic programming: an introduction*, volume 1. Morgan Kaufmann San Francisco.

Barlow, M., Galloway, J., and Abbass, H. A. (2002). Mining evolution through visualization. In *ALife VIII workshops*, pages 103–112. Citeseer.

Bourque, P., Fairley, R. E., et al. (2014). *Guide to the Software Engineering Body of Knowledge (SWEBOK (R)): Version 3.0*. IEEE Computer Society Press.

Bugzilla (2015). Bug 36854 – [list] if list-style-position is inside, bullet takes own line. `https://bugzilla.mozilla.org/show_bug.cgi?id=36854`. (Visited on 06/09/2015).

DeMarco, F., Xuan, J., Le Berre, D., and Monperrus, M. (2014). Automatic repair of buggy if conditions and missing preconditions with smt. In *Proceedings of the 6th International Workshop on Constraints in Software Testing, Verification, and Analysis*, pages 30–39. ACM.

Eisenmann, J., Lewis, M., and Parent, R. (2013). Trace selection for interactive evolutionary algorithms. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 327–334. ACM.

Harman, M. and Jones, B. F. (2001). Search-based software engineering. *Information and Software Technology*, 43(14):833–839.

Langdon, W. B. and Harman, M. (2013). Optimising existing software with genetic programming. *IEEE Transactions on Evolutionary Computation*, (99).

Le Goues, C., Dewey-Vogt, M., Forrest, S., and Weimer, W. (2012a). A systematic study of automated program repair: Fixing 55 out of 105 bugs for $8 each. In *Software Engineering (ICSE), 2012 34th International Conference on*, pages 3–13. IEEE.

Le Goues, C., Forrest, S., and Weimer, W. (2013). Current challenges in automatic software repair. *Software Quality Journal*, 21(3):421–443.

Le Goues, C., Nguyen, T., Forrest, S., and Weimer, W. (2012b). Genprog: A generic method for automatic software repair. *Software Engineering, IEEE Transactions on*, 38(1):54–72.

Leveson, N. G. and Turner, C. S. (1993). An investigation of the therac-25 accidents. *Computer*, 26(7):18–41.

Liao, Y.-H. and Sun, C.-T. (2001). An educational genetic algorithms learning tool. *IEEE Trans. Education*, 44(2):20.

Lions, J.-L. et al. (1996). Ariane 5 flight 501 failure.

Martinez, M. and Monperrus, M. (2013). Mining Software Repair Models for Reasoning on the Search Space of Automated Program Fixing. *Empirical Software Engineering*, Online First. Accepted for publication on Sep. 11, 2013.

Perkins, J. H., Kim, S., Larsen, S., Amarasinghe, S., Bachrach, J., Carbin, M., Pacheco, C., Sherwood, F., Sidiroglou, S., Sullivan, G., et al. (2009). Automatically patching errors in deployed software. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 87–102. ACM.

Pohlheim, H. (1999). Visualization of evolutionary algorithms-set of standard techniques and multidimensional visualization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pages 533–540. San Francisco, CA.

Pohlheim, H. (2015). Geatbx - genetic and evolutionary algorithm toolbox for matlab. `http://www.geatbx.com/`. (Visited on 06/24/2015).

Pressman, R. S. (2005). *Software engineering: a practitioner's approach*. Palgrave Macmillan.

Weimer, W., Forrest, S., Le Goues, C., and Nguyen, T. (2010). Automatic program repair with evolutionary computation. *Communications of the ACM*, 53(5):109–116.

Weimer, W., Nguyen, T., Le Goues, C., and Forrest, S. (2009). Automatically finding patches using genetic programming. In *Proceedings of the 31st International Conference on Software Engineering*, pages 364–374. IEEE Computer Society.

Yin, Z., Yuan, D., Zhou, Y., Pasupathy, S., and Bairavasundaram, L. (2011). How do fixes become bugs? In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, pages 26–36. ACM.

Zhang, Y. (2010). *Multi-Objective Search-based Requirements Selection and Optimisation*. PhD thesis, University of London.

Zhivich, M. and Cunningham, R. K. (2009). The real cost of software errors.