Geração de Bases de Dados de Teste pela Aplicação de Programação Evolucionária

Filipe Alves de Almeida[1], Plínio de Sá Leitão-Júnior[1], Auri Marcelo Rizzo Vincenzi[1], Fábio Nogueira de Lucena[1]

[1]Instituto de Informática – Universidade Federal de Goiás (UFG)
74.001-970 – Goiânia – GO – Brazil

psy-filipe@hotmail.com, {plinio,auri,fabio}@inf.ufg.br

Abstract. *This paper presents an approach to SQL instruction testing via Mutation Analysis that uses Evolutionary Programming (EP) to select data to be used in the assessment of mutants. Based on a heuristic perspective, our aim is to select an effective data set which may help detect faults in the SQL instructions of a given application. Results were obtained from experiments that compare evolutionary programming, random generation and genetic algorithm. Evolutionary programming reached final values earlier than Genetic Algorithm at 2/3 of test cases, which can be an indication of lower relative cost, and both are superior to detect defects in relation to generating random data.*

Resumo. *Este artigo apresenta uma abordagem para teste de instrução SQL através de Análise de Mutantes que usa a Programação Evolucionária (PE) para selecionar os dados a serem utilizados na avaliação de mutantes. Baseada em uma perspectiva heurística, o nosso objetivo é selecionar um conjunto de dados eficaz, que pode ajudar a detectar defeitos nas instruções SQL de uma determinada aplicação. Os resultados foram obtidos a partir de experimentos que comparam Programação Evolucionária, Geração Aleatória e Algoritmo Genético. Programação evolucionária atingiu valores finais antes do algoritmo genético em 2/3 dos casos de teste, o que é uma indicação de custo relativamente baixo, e ambos são superiores para detectar defeitos em relação à geração de dados aleatórios.*

1. Introdução

Teste é o processo de executar um programa ou sistema com a finalidade de encontrar defeitos não detectados durante a fase de desenvolvimento. A atividade de teste em si possui incertezas; por exemplo, o fato de somente um subconjunto de elementos do domínio de entrada ser selecionado para o teste introduz uma taxa de incerteza em relação se (todos) os defeitos serão revelados.

O contexto desta pesquisa refere-se às incertezas inerentes ao teste de software, especificamente durante o planejamento e a aplicação do teste, em atividades tais como: seleção de dados de teste, execução do teste e checagem de resultados. Especificamente, o trabalho aborda teste do software que envolve persistência de dados, intitulado Aplicação de Banco de Dados (ABD). ABDs fazem uso de algum tipo de linguagem para acesso aos dados, onde se destaca a SQL (*Structured Query Language*).

O problema em questão é "como gerar bases de dados para o teste de aplicações de banco de dados a um baixo custo e com eficácia para a descoberta de defeitos em

instruções SQL?". Os dados de teste para essa classe de aplicações são bases de dados, que serão "lidas" pelo software em teste, o que denota elevada complexidade do problema, devido ao extenso conjunto de possibilidades para a geração dessas bases de dados.

A geração de bases de dados para o teste de ABDs pode, em geral, ser representada por um problema de otimização ou busca. Assim, é possível obter resultados importantes para dados de teste, aplicando-se técnicas de busca, com um custo computacional relativamente baixo, apesar da complexidade do problema em si.

Computação Evolucionária (CE) é um paradigma de Inteligência Computacional (IC), que busca reproduzir processos naturais de evolução, onde é usado o conceito de sobrevivência (Jong, 2006). Como os processos e os produtos de engenharia de software são dependentes de decisões humanas, o uso de técnicas de CE pode agregar qualidade à análise, ao projeto, à implementação, ao teste e à manutenção de software.

Uma das abordagens que pode ser usada para aumentar a eficácia dos testes de instruções SQL é a Análise de Mutantes (Derezinska, 2007), que objetiva mensurar a qualidade de um conjunto de bases de entrada. Para tal, são produzidas várias versões da instrução SQL, denominadas mutantes, cada uma com uma pequena modificação na sintaxe da instrução original. Um conjunto de bases de entrada é promissor para revelar defeitos se o resultado obtido pela execução da instrução original difere dos resultados dos seus mutantes. Ou seja, o conjunto de bases de entrada foi capaz de detectar a diferença entre a instrução original e os seus mutantes. Essa técnica vem sendo utilizada como uma evidência de que os dados de teste são reveladores de defeitos.

1.1 Objetivos

O objetivo principal do artigo é testar Aplicações de Banco de Dados, visando à descoberta de potenciais defeitos em instruções SQL, aplicando-se: (i) conceitos de Computação Evolucionária para a geração de bases de entrada; e (ii) Análise de Mutantes para medir a qualidade do conjunto de bases geradas.

O objetivo principal da pesquisa pode ser decomposto em:

1. Representar bases de dados como indivíduos que podem ser criados e evoluídos por meio da aplicação de algoritmos da computação evolucionária.

2. Desenvolver algoritmos e suporte computacional para a aplicação da Computação Evolucionária na geração de bases de teste de ABDs, segundo a meta-heurística Programação Evolucionária.

3. Aplicar Análise de Mutantes para avaliar e comparar bases de dados de teste geradas: (i) pelo emprego da meta-heurística Programação Evolucionária (PE); (ii) pelo uso de Algoritmos Genéticos (AG); e (iii) aleatoriamente.

Organização do artigo: a Seção 2 apresenta os trabalhos correlatos; a Seção 3 introduz análise de mutantes e o problema da geração de dados de teste; a Seção 4 apresenta a Programação Evolucionária; a representação cromossômica e a base de produção são, respectivamente, apresentadas nas Seções 5 e 6; o desenvolvimento e a obtenção de resultados são explorados nas Seções 7 e 8, respectivamente; a Seção 9 analisa resultados; e a Seção 10 apresenta conclusões, contribuições e desdobramentos.

2. Trabalhos Correlatos

Algumas áreas exploradas e correlatas são: (i) redução de mutantes para obter novos casos de teste que melhoram a qualidade de uma suíte de teste inicial (Domínguez-Jiménez et al., 2011); (ii) aplicação de meta-heurísticas para a geração automática de dados de teste (McMinn, 2004); (iii) geração de mutantes para instruções SQL (Tuya et al., 2006, Tuya et al., 2007); (iv) teste de injeção SQL (Kosuga et al., 2007); e (v) teste de cobertura de instruções SQL (Suárez-Cabal and Tuya, 2004). Colanzi et al. (2012) apresentam uma revisão sobre técnicas de busca aplicadas à engenharia de software.

Por se tratar de um problema em que computação evolucionária foi pouco explorada em sua solução, identificou-se na literatura um único trabalho, (Monção et al., 2013), que busca reduzir bases de dados de produção na perspectiva do teste de aplicações SQL. A ideia do artigo é produzir bases de dados menores, com menor custo associado, sem perdas de eficácia significantes segundo à análise de mutantes. São empregadas as meta-heurísticas Algoritmo Genético e Fertilização In Vitro com resultados promissores. Não há referência à meta-heurística Programação Evolucionária.

3. Análise de Mutantes e o Problema da Geração de Dados de Teste

Um dos grandes desafios da Engenharia de Software é a automatização do processo de teste de software, e uma parte crucial deste processo é a geração de casos de teste. O objetivo da automatização da geração de dados de teste é reduzir o custo e otimizar o processo através da obtenção de um conjunto de casos de teste de qualidade; ou seja, realizar de forma automática a cobertura satisfatória do software em teste, e revelar assim um maior número de erros e inconsistências.

Uma estratégia que vem ganhando muita atenção é a aplicação de geração de dados de teste através de algoritmos de busca baseados em meta-heurísticas. A abordagem é justificada pela natureza do problema: selecionar dados reveladores de defeitos a partir de um domínio (grande) de entrada. Ou seja, o espaço de busca é o conjunto de todos os dados de teste possíveis, e o objetivo da busca é a obtenção de um conjunto reduzido de dados que otimize o processo de teste.

A computação evolucionária utiliza o conceito de evolução de uma população, a qual é composta por indivíduos candidatos a solucionar um dado problema. Meta-heurísticas são empregadas no processo de evolução e os indivíduos representam Bases de Dados de Teste (BDT), que serão usadas para o teste de instruções escritas em SQL.

Um grupo de indivíduos é selecionado a cada geração dos resultados produzidos pela execução dos algoritmos pertinentes à Programação Evolucionária e ao Algoritmo Genético. Essa seleção é baseada na qualidade de cada indivíduo, atestada pela análise dos comandos caracterizados como mutantes da instrução SQL original em teste.

A princípio, o resultado da execução da instrução SQL original (que gera os mutantes) é comparado com o resultado da execução de uma instrução mutante. Esse processo é realizado para todos os mutantes e com a mesma instrução original. O valor da aptidão é calculado através da divisão da quantidade de mutantes mortos (Q) pela subtração entre a quantidade total de mutantes comparados (T) e a quantidade de mutantes equivalentes (E). Ou seja, a aptidão do indivíduo (*fitness*) é igual a Q/(T-E). Quanto mais próximo o valor for de 1, melhor será o indivíduo. Se o indivíduo não é capaz de matar nenhum mutante, atribui-se a ele um *fitness* igual a zero.

4. A Meta-heurística Programação Evolucionária

A meta-heurística estudada nesse trabalho é a Programação Evolucionária (PE) (Fogel, 1962). A característica principal da PE que a diferencia das demais técnicas evolutivas, tais como Algoritmos Genéticos e Programação Genética, é o fato de que a técnica não foca no desenvolvimento de modelos genéticos, como as outras técnicas citadas, e sim em modelos comportamentais. Dessa forma, somente é considerada a evolução fenotípica, sendo aplicados apenas operadores de mutação e seleção, ou seja, não há a recombinação de elementos, o que caracterizaria a evolução genotípica.

Os quatro componentes principais da abordagem de um problema pela Programação Evolucionária, exibidos na Figura 1, são: **inicialização**, **mutação**, **avaliação** e **seleção**. Semelhante a outras técnicas da CE, o primeiro passo da aplicação da meta-heurística se dá pela geração aleatória de uma população inicial de indivíduos, que potencialmente cobrem de forma uniforme o espaço de busca do problema.

Em seguida, é introduzida na população uma variação, através de operadores de mutação aplicados aos indivíduos, que geram indivíduos filhos. A etapa de avaliação é baseada em uma função *fitness* relativa, obtida para cada indivíduo através da comparação com um grupo de outros indivíduos selecionados aleatoriamente.

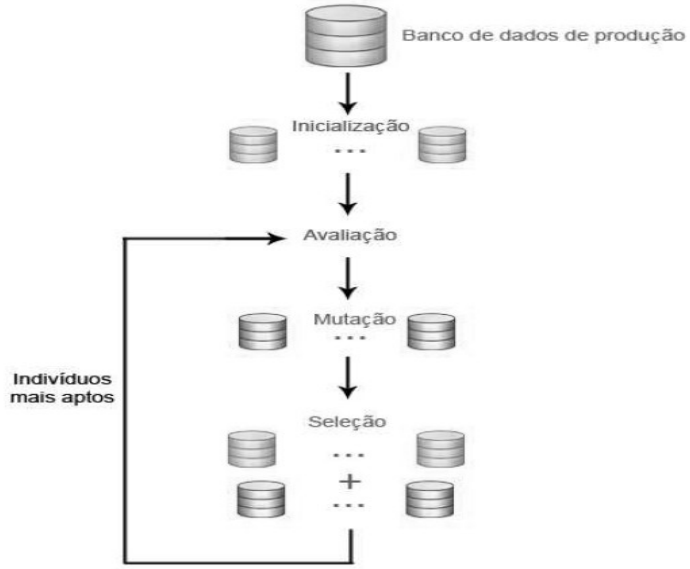

Figura 1. Fluxo em que se usa a meta-heurística Programação Evolucionária.

Dessa forma, a busca não é conduzida por uma função *fitness* absoluta, mas sim pelo “erro comportamental” de cada indivíduo. Após a avaliação de todos os indivíduos, ocorre uma competição entre pais e filhos pela sobrevivência para a próxima geração.

Diversos operadores de mutação podem ser usados, e tais operadores estão fortemente acoplados à representação do problema. Mas a questão principal que deve ser mantida em mente é que na PE a mutação é a única forma de introdução de variação na população. Os operadores devem ser bem definidos, para que haja um balanço adequado entre a busca global e busca local (*exploration-exploitation*), permitindo uma exploração vasta do espaço de busca inicialmente, seguido do aperfeiçoamento dos indivíduos, ou seja, uma busca local, para então obter-se um resultado otimizado.

Existem também vários operadores de seleção a serem utilizados, o que depende também fortemente da representação do problema e do indivíduo. Diversos métodos de

seleção podem ser aplicados, entre os quais, os mais comuns são: elitismo (selecionar melhores indivíduos da população), seleção por torneio (selecionar melhores indivíduos de subgrupos aleatórios da população), por roleta (a chance de um indivíduo ser selecionado é proporcional à sua adequação) ou ranking (a chance de um indivíduo ser selecionado é proporcional à sua posição em relação ao resto da população).

5. Representação Cromossômica

No contexto deste trabalho, foi adotada a representação cromossômica definida em (Monção et al., 2013). Um indivíduo (cromossomo) é composto pela identificação das suas *tuplas*. Cada *tupla* é representada por dois números inteiros: um sendo o campo identificador único na tabela em que se encontra, e outro indicando em qual tabela está. O indivíduo, portanto, é representado por um conjunto desses pares de valores, ou seja, por um conjunto de tuplas que determina um banco de dados de teste (BDT).

Um exemplo de representação, ilustrada na Figura 2, corresponde ao caso em que um indivíduo é um BDT contendo 5 tuplas da Tabela **Department**, cujo identificador único das *tuplas* é o campo **Dnumber**. Nesse caso, cada gene (*tupla*) é composto por um par de valores, sendo que o primeiro número indica o identificador único da *tupla* na tabela, ou seja um valor para **Dnumber**, e o outro valor indica a tabela que contém a *tupla*, nesse caso o valor 2 corresponde à Tabela **Department**.

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Management	10	333445555	1995-05-22
Division	11	666778888	1992-11-05
Sector	12	222334444	1996-01-30
Board	13	555667777	1995-05-22
Agency	14	444556666	1997-08-16

INDIVÍDUO

10, 2	11, 2	12, 2	13, 2	14, 2

Figura 2. Exemplo de indivíduo segundo à representação cromossômica.

6. Base de Dados de Produção

Um Banco de Dados de Produção (BDP) é usado como espaço de busca para gerar bases de dados de teste, tendo sido criado a partir do Modelo Conceitual *Company* definido em (Elmasri e Navathe, 2011), cujo esquema é mostrado na Figura 3.

Nas tabelas onde a chave primária é composta, foi adicionado um campo (**id**) para armazenar um valor sequencial único para cada tupla da tabela, permitindo assim a identificação de cada tupla usando apenas um campo. Para popular o banco foi utilizado o software *Spawner Data Generator* (Grover, 2013), que pode ser configurado para gerar arquivos SQL contendo instruções de inserção com diferentes tipos de dados aleatórios para cada tabela, sempre respeitando as restrições de integridade do banco.

As tabelas foram preenchidas com as seguintes quantidades de *tuplas*: **Employee** (100.000), **Department** (1.000), **Project** (5.000), **Dependent** (50.000),

Works_on (75.000) e **Dept_locations** (10.000). Com essa diversidade e quantidade de dados, é possível abranger muitas situações reais existentes em ABDs, em que os testes são, de fato, importantes para garantir uma melhor qualidade do software.

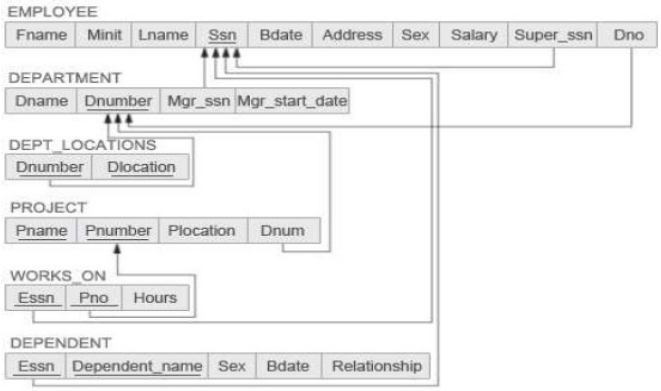

Figura 3. Esquema do banco de dados Empresa.

7. Desenvolvimento

O projeto do suporte computacional para o emprego e avaliação da meta-heurística Programação Evolucionária no contexto de teste de ABDs envolve algumas questões importantes que influenciam a execução do algoritmo.

Uma dessas questões diz respeito à representação cromossômica definida. Um caso de grande importância, que deve ser considerado de forma especial nessa representação, é o chamado problema de infactibilidade do BDT. Por exemplo, esse problema pode ocorrer quando há a dependência entre tuplas devido à integridade referencial: se alguma das *tuplas* dependentes não estiver na base de dados, então há uma inconsistência dos dados. Essa situação impede que o BDT seja considerado um banco de dados consistente e sua aplicação à atividade de teste é prejudicada.

No contexto do trabalho atual, utilizaram-se comandos SQL que referenciam tuplas pertencentes a uma única tabela. Essa redução de escopo evitou o problema de infactibilidade de indivíduo acima descrito, deixando o seu tratamento para os esforços futuros da pesquisa. A tabela escolhida foi **Employee**, a qual possui 100.000 *tuplas*, e cada indivíduo possuirá um subconjunto das tuplas dessa tabela.

Outra questão importante ligada ao projeto é a necessidade de armazenar informações adicionais sobre as instruções (original e mutantes), bem como a relação de quais mutantes foram mortos por determinado indivíduo ao longo da execução do algoritmo. Para lidar com isso, foi criado um Banco de Dados de Mutantes (BDM), cujo esquema é ilustrado na Figura 4. O BDM também armazena outras informações sobre os indivíduos criados na execução, tais como a geração em que ele nasceu e o seu *fitness*.

Existem várias maneiras de se definir os operadores de mutação e cruzamento, sendo importante especificar o funcionamento de cada um no projeto da implementação. A mutação, presente na Programação Evolucionária e no Algoritmo Genético, ocorre por gene com uma determinada probabilidade, geralmente baixa. O cruzamento, presente apenas no Algoritmo Genético, funciona com ponto de corte aleatório, sendo que os indivíduos participantes do cruzamento são selecionados por torneio.

As instruções mutantes foram obtidas pela ferramenta **SQLMutation** (Tuya et al., 2006), as quais representam pequenas modificações aplicadas ao comando SQL original; por exemplo, a remoção ou a inserção de operadores que visam a encontrar erros de lógica ou envolver manipulação de valores nulos.

A implementação do projeto foi feita usando a linguagem de programação **Java** e sistema gerenciador de banco de dados **Mysql**. A execução do algoritmo da meta-heurística PE foi realizada através dessa implementação, cujo modelo de classes está representado na Figura 5.

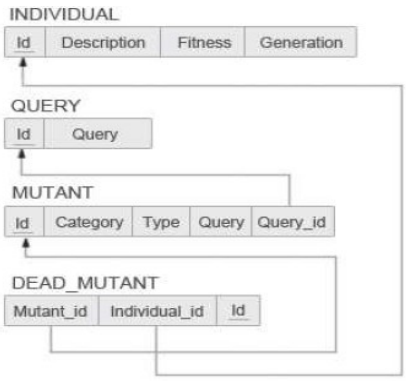

Figura 4. Esquema do banco de dados de mutantes.

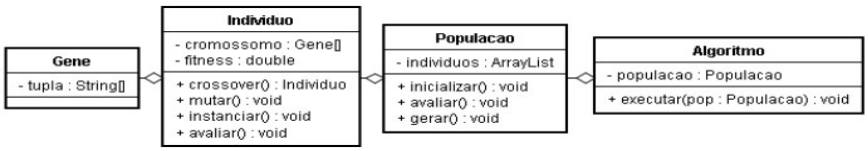

Figura 5. Diagrama de classes utilizadas na implementação.

8. Parametrização e Resultados Obtidos

O algoritmo inicialmente gera uma população aleatória de 50 indivíduos, a partir do banco de dados de produção *Company*. Essa geração aleatória consiste em selecionar as tuplas da Tabela **Employee** que irão compor o indivíduo, no total de 100. Em seguida, cada indivíduo é avaliado através da análise de mutantes, para atribuir o valor do *fitness*.

Após isso, novas populações serão definidas ao longo das 25 gerações através da seleção competitiva entre a população atual e os seus filhos (resultantes de uma variação a uma probabilidade de 3% de mutação gene-a-gene, isto é, *tupla*-a-*tupla*).

De forma semelhante, o Algoritmo Genético foi executado com um operador de cruzamento, em adição ao operador de mutação. Novos indivíduos são gerados através de um cruzamento, entre um indivíduo pai e um indivíduo mãe da população atual da geração. O cruzamento ocorre através de um ponto de corte aleatório. A cada geração, são mantidos os dois melhores indivíduos, caracterizando uma seleção elitista.

A instrução SQL utilizada na análise empírica foi: “select T.sex, max(T.salary), min(T.salary), avg(T.salary) from (select ssn, fname, sex, salary from employee where (salary >= 800 and salary < 10000 and sex = "M") or (salary > 10000 and sex = "F") or (salary > (0.7*(select avg(salary) from employee where address not like "%- SP%")))) T group by T.sex”.

A geração de mutantes criou 250 novas instruções a partir da instrução original. Definida a parametrização dos algoritmos, foram realizados três experimentos, visando à redução de ameaças à validade dos resultados. Os dados obtidos com as execuções do Algoritmo Genético e da Programação Evolucionária estão ilustrados na Tabela 1, em

que são exibidos: (i) o valor do *fitness* do melhor indivíduo encontrado no experimento; (ii) a geração em que ele foi gerado inicialmente, ou seja, a geração que atinge o melhor valor; e (iii) a evolução na quantidade de mutantes mortos de cada experimento, isto é, a quantidade de mutantes mortos no uso da meta-heurística, que não seriam mortos em casos aleatórios sem a sua aplicação.

Tabela 1. Resultados obtidos com execução dos algoritmos.

Experimento	Melhor *fitness*	Geração	Mutantes	Melhor *fitness*	Geração	Mutantes
	Algoritmo Genético			Programação Evolucionária		
1	0.7230	11	5	0.6906	13	4
2	0.7194	22	5	0.6978	7	5
3	0.7158	21	7	0.6906	10	4

9. Análise de Resultados

O *fitness* obtido usando o BDP é 0,7410, e revela que o banco formado por toda a tabela **Employee** é capaz de matar 74% dos mutantes, ou seja, aproximadamente 186 instruções mutantes de um total de 250. O objetivo da geração de bases de teste é aproximar-se desse valor com bases de dados menores em relação ao BDP, pois representa uma redução de custo para a atividade de teste.

Os resultados obtidos na execução da PE mostram que o melhor *fitness* encontrado em cada experimento converge para um valor próximo a 0,70 (por volta de 175 mutantes são mortos pelo melhor indivíduo gerado em cada execução). Além disso, cerca de 2% das instruções mutantes, ou seja, cinco mutantes aproximadamente, não seriam mortos em casos aleatórios sem a aplicação de algum algoritmo evolucionário. Essa variação existente entre mutantes que são mortos gerando bases de dados aleatoriamente e mutantes que são mortos através da execução de alguma meta-heurística, demonstra e comprova a capacidade de algoritmos como a PE e AG para resolver problemas de otimização ligados à geração de bases de teste para ABDs.

A Programação Evolucionária não faz uso do operador de cruzamento e, por conta disso, o operador de mutação por si só não se mostra capaz de introduzir uma variação positiva que leve a melhores soluções e favoreça a evolução em relação ao Algoritmo Genético. A introdução de variação gene-a-gene não se mostra capaz de fornecer uma mutação adequada ao tipo de indivíduo do problema tratado. Por isso, para que haja um balanço adequado entre a busca global e a busca local (*exploration-exploitation*), é necessário definir operadores de mutação específicos para o indivíduo no problema de geração de dados de teste para ABDs.

Assim, os resultados do AG foram relativamente melhores quando comparados aos da PE. Percebe-se que no caso do AG, houve um melhor balanço entre a busca global e a busca local, permitindo melhor exploração do domínio de busca e obtenção de melhores resultados.

A análise dos resultados está sendo feita sobre o melhor indivíduo de cada experimento. Outra opção seria realizar a análise de mutantes sobre todo o conjunto da população. Como os indivíduos são bem pequenos comparados ao BDP, essa estratégia pode ser vantajosa, pois o resultado da união das tuplas de toda a população pode ter um resultado melhor que o da análise apenas do melhor indivíduo.

A parametrização utilizada nas execuções reflete diretamente no que foi obtido com os experimentos. A definição da quantidade de gerações, o tamanho do indivíduo,

a probabilidade de mutação, a instrução SQL utilizada, entre outros, são características que contribuem para a obtenção dos resultados. No contexto do problema de geração de bases de teste, a configuração tal como foi definida mostrou-se capaz de revelar boas soluções, não impedindo possíveis modificações para futuros experimentos.

10. Conclusões

Nesse trabalho, foi possível observar que a aplicação da Computação Evolucionária aliada à Análise de Mutantes é promissora para resolver o problema da Geração de Bases de Teste para a descoberta de defeitos em instruções SQL presentes em Aplicações de Banco de Dados.

Especificamente, a meta-heurística Programação Evolucionária foi empregada no âmbito da evolução de indivíduos candidatos a dados de entrada para o teste de instruções SQL.

A análise empírica explorou a Análise de Mutantes como medida de qualidade dos dados de teste e empregou Programação Evolucionária (PE), Algoritmo Genético (AG) e Geração Aleatória (GA) na seleção de dados de teste, obtendo os seguintes resultados: (i) PE e AG são superiores para detecção de defeitos em relação à geração aleatória de dados; e (ii) AG foi superior a PE em termos de qualidade, contudo em 2/3 dos casos PE alcançou valores finais de forma mais precoce, o que pode ser um indício de menor custo relativo.

10.1 Contribuições

As seguintes contribuições podem ser realçadas:

- esforço para aplicar meta-heurísticas ao problema da geração de dados para o teste de instruções SQL; nesse sentido, o artigo demonstra todas as etapas de um processo, que inclui: criação de base de produção, geração aleatória de bases de teste iniciais, representação cromossômica, aplicação de meta-heurística para a evolução de população no contexto do teste instruções SQL e análise de resultados;

- maturação da representação cromossômica para o problema, a qual foi definida em (Monção et al., 2013), pertinente à seleção de tuplas de bases de produção, visando à redução do custo do teste pela diminuição da base de dados de entrada; por se tratar de um problema ainda não muito explorado na literatura através de meta-heurísticas evolucionárias, essa representação cromossômica representa um passo importante, reforçada pelos resultados promissores;

- resultados promissores apontam (abrem o caminho) para a possibilidade do emprego de outras meta-heurísticas ao problema;

- outras contribuições, tais como desenvolvimento de suporte computacional e construção da base de produção, que caracteriza o espaço de busca.

10.2 Desdobramentos futuros

Alguns desdobramentos futuros para a pesquisa são: (i) definição de operadores de mutação pertinentes à Programação Evolucionária, específicos à solução do problema de geração de dados de teste para ABDs; (ii) utilização de novos parâmetros para a análise empírica, tais como tamanho do indivíduo e emprego de medida de qualidade no conjunto da população (em vez de cada indivíduo); (iii) extensão do estudo a outras

meta-heurísticas evolucionárias; e (iv) expansão do escopo do problema, pela utilização de várias tabelas do banco de dados e pelo tratamento da infactibilidade de indivíduos.

Referências

Colanzi, T. E., , Vergilio, S. R., Assunção, W. K. G., Pozo, A. (2013) Search Based Software Engineering: Review and analysis of the field in Brazil, The Journal of Systems and Software, 86:970– 984.

Derezinska, A. (2007) An experimental case study to applying mutation analysis for SQL queries. IMCSIT.

Domínguez-Jiménez, J.J., Estero-Botaro , A., García-Domínguez, A., Medina-Bulo, I. (2011) Evolutionary mutation testing. Information and Software Technology, 53:10.

Elmasri, R. and Navathe, S.B. (2011) Fundamentals of Database Systems, Sixth Edition. Addison Wesley.

Spawner Data Generator, http://sourceforge.net/projects/spawner/. Último acesso em Março de 2013.

L.J. Fogel. (1962) Autonomous automata, Industrial Research, 4:14-19.

Jong, K. A. D. (2006) Evolutionary Computation A Unified Approach. Massachusetts Institute of Technology, Cambridge, MA.

Kosuga, Y., Kono, K., Hanaoka, M., Hishiyama, M., Takahama, Y. (2007), Sania: Syntactic and Semantic Analysis for Automated Testing against SQL Injection, Computer Security Applications Conference (ACSAC).

McMinn, P. (2004), Search-based software test data generation: a survey. Journal of Software Testing, Verification and Reliability, 14:2.

Monção, A. C. B. L., Camilo-Junior, C. G., Queiroz, L. T., Rodrigues, C. L. Leitão-Junior, P. S., Vincenzi, A. M. R. (2013) Shrinking a Database to Perform SQL Mutation Tests Using an Evolutionary Algorithm, Congress on Evolutionary Computation (CEC).

Suárez-Cabal, M.J., Tuya J. (2004), Using an SQL coverage measurement for testing database applications, ACM SIGSOFT Software Engineering Notes, 29:6.

Tuya, J., Suárez-Cabal, M.J. and De Lariva C. (2006), SQLMutation: A tool to generate mutants of SQL database queries. Second Workshop on Mutation Analysis.

Tuya, J., Suárez-Cabal, M.J. and De Lariva C. (2007), Mutating database queries. Information and Software Technology, 49(4):398 – 417.