Applying Genetic Algorithms to Data Selection for SQL Mutation Analysis

Ana C.L. Monção
Institute of Informatics
Federal University of Goias
acblmoncao@gmail.com

Celso G. Camilo-Jr
Institute of Informatics
Federal University of Goias
celso@inf.ufg.br

Leonardo T. Queiroz
Institute of Informatics
Federal University of Goias
leonardo.queiroz@gmail.com

Cássio L. Rodrigues
Institute of Informatics
Federal University of Goias
cassio@inf.ufg.br

Plínio de Sá Leitão-Jr
Institute of Informatics
Federal University of Goias
plinio@inf.ufg.br

Auri M.R. Vincenzi
Institute of Informatics
Federal University of Goias
auri@inf.ufg.br

ABSTRACT

This paper presents an approach to Structured Query Language (SQL) instruction tests via Mutation Analysis that uses Evolutionary Algorithms (GA) 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. The results obtained from experiments reveal a good performance using GA metaheuristic.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging—*Testing tools (e.g., data generators, coverage testing)*; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Heuristic methods*

Keywords

Software Testing; Database Application; SQL Mutation Analysis; Data Selection; Genetic Algorithms.

1. INTRODUCTION

Being, SQL instructions, crucial components in applications which use relational databases, is important to find a systematic approach to run tests and ensure the coverage required to identify most faults. The use of the same production environment for testing represents a serious risk to the applications running. However, there is great difficulty in finding or generating data to perform the tests with behave similar to a production database, as well as a significant amount of information.

The aim of this work is to select an adequated subset of tuples, with reduced input domain and reasonable computational and operational costs, from a real data base to be used as a test database for SQL instructions tests. Hence, we intend to apply the principles of Evolutionary Computation that contribute to the selection of an effective test data set, and to measure the adequacy of each subset selected, we make use of the results of SQL Mutation Analysis application.

GECCO'13 Companion, July 6–10, 2013, Amsterdam, The Netherlands.
ACM 978-1-4503-1964-5/13/07.

2. SQL MUTATION ANALYSIS

Mutation Analysis is a criterion of fault-based testing which involves inserting small syntactic changes into the code of testing product P, thus generating its mutants P'. These mutants are generated from mutation operators that cause syntactic changes based on the most common mistakes made by programmers regarding the language of the program under testing [1].

A subsequent analysis verifies whether such changes are noticeable by test set T, which compares the result from the original program with that of the mutant. If results are different, the mutant P' is said to have been killed by T. The greater the number of mutants killed, the higher the quality of T [1]. This is known as score mutation and is calculated as follows:

$$ms(P,T) = \frac{DM(P,T)}{M(P) - EM(P)}$$

In which: $DM(P,T)$ is the number of mutants killed by test cases in T; $M(P)$ is the total number of generated mutants; $EM(P)$ is the number of mutants equivalent to P.

Tuya et al. [4] proposed a set of SQL operators to generate mutants that were implemented by *SQLMutation* tool [3] used in this work to generate mutants.

3. USING GENETIC ALGORITHMS FOR DATA SELECTION

Having a ***PDB*** (Production Database), we need to select a minimum subset of tuples to build a Test Database (***TDB***) to be used as an input data set to SQL instructions tests. The selection process resorts to Genetic Algorithms (GA) in an attempt to find, heuristically, an effective data set that is capable of detecting the majority of faults in an application's SQL instructions. To use this approach, we choose a representation model that uses a chromosome implemented as a vector, in which each position represents a tuple from one of the *PDB* tables.

4. EXPERIMENTS AND RESULTS

We created an environment to simulate a real production database (*PDB*) based on the model COMPANY proposed by Elmasri and Navathe's [2]. We selected the table ***employee***, loaded with 100.000 tuples, for the creation of instructions. A set of 30 SQL instructions was created with

various levels of complexity. Mutants were generated for each instruction through the *SQLMutation* tool[3].

From these 30 SQL instructions, 3 were selected considering the highest coverage of mutation operators and complexity. Each one was assessed by Mutation Analysis in order to calculate the mutation score, having the entire *PDB* as input data.

For these, we performed 1.000 random selection experiments with 100 tuples each one (1.000 *TDB*s) and 10 random selection experiments with 10.000 tuples (10 *TDB*s).

Attempting to find better results (scores) or similar results with less tuples we performed the first experiments using GA with the following parameters: 30 generations each one with 100 individuals of 100 genes; 100% of crossover rate with random cut-off; 3% of mutation rate per gene and elitism of the two best individuals.

In Table 1 and Figure 1 we may see better results with the use of GA. For instruction 6 we achieved, more than once, the same mutation score as one achieved by *PDB* with only 0,1 % of the amount of data. For instructions number 10 and 11, the results did not reach the mutation score of *PDB* but they were much better than randomized experiments.

5. REFERENCES

[1] E. F. Barbosa, J. C. Maldonado, and A. M. R. Vincenzi. Introduction to software testing.

[2] E. Ramez and S. B. N. *Fundamentals of Database Systems*. Pearson, 6ł edition, 2005.

[3] J. Tuya, M. J. Suarez-Cabal, and C. de la Riva. Sqlmutation: A tool to generate mutants of sql database queries. *Mutation Analysis, Workshop on*, 0:1, 2006.

[4] J. Tuya, M. J. Suárez-Cabal, and C. de la Riva. Mutating database queries. *Information and Software Technology*, 49(4):398 – 417, 2007.

Table 1: Random and GA results

SQL	*PDB*	Random (100 tuples)			Random (1000 tuples)			GA (100 tuples)		
		Best	Average	Deviation	Best	Average	Deviation	Best	Average	Deviation
6	0,844	0,7778	0,1626	0,0648	0,8000	0,2760	0,1123	0,844	0,5755	0,2035
10	0,928	0,5381	0,0756	0,0523	0,6398	0,2287	0,1200	0,8390	0,7907	0,0440
11	0,939	0,4763	0,0635	0,0430	0,5675	0,1680	0,1097	0,8074	0,7581	0,0357

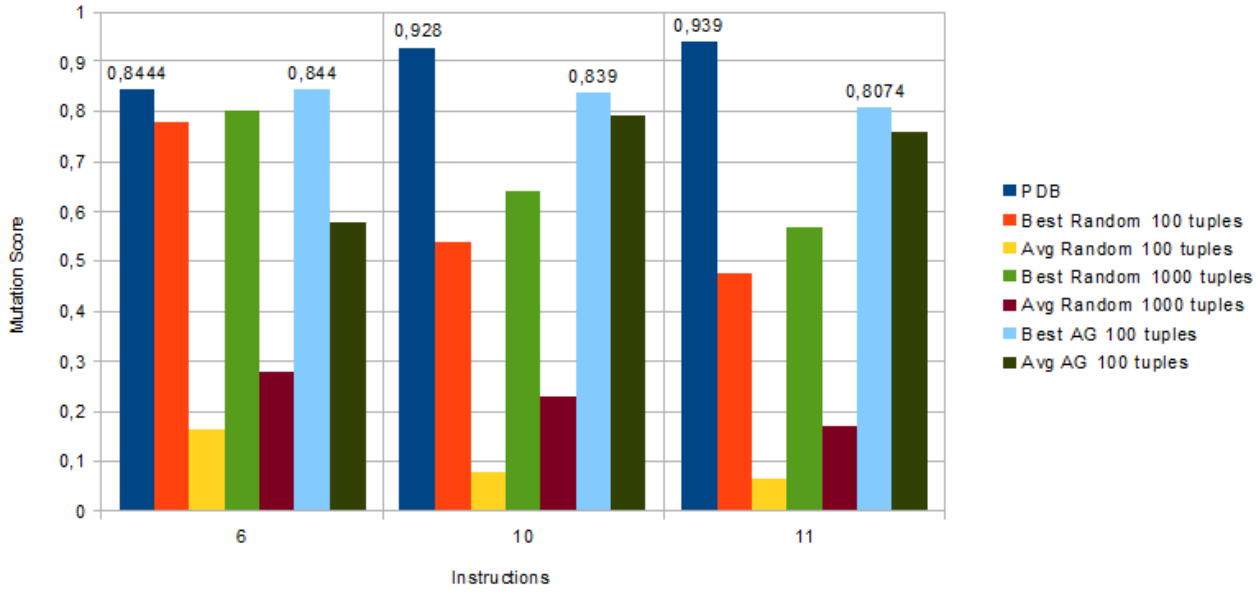

Figure 1: Scores of random and GA experiments