Automated Software Engineering manuscript No.
(will be inserted by the editor)

Empirically Evaluating the Efficiency of Search-based Test Data Generation for Relational Database Schemas

Cody Kinneer · Luke Smith · Gregory Kapfhammer · Chris Wright · Phil McMinn

Received: date / Accepted: date

Abstract When evaluating an algorithm, it is often useful to speak of it's efficiency in terms of it's worst-case complexity. However, for certain cases such as search-based algorithms, determining an algorithm's efficiency by theoretical analysis is difficult, and has not been reported in the literature. This paper introduces a framework for conducting automated empirical studies of algorithms by doubling the size of the input and observing the change in execution time. We apply this method to the domain of data generation for relational database schemas. After implementing a technique for systematically doubling the size of schemas, we conduct an empirical study on the search-based data generation tool *SchemaAnalyst*. For the parameters of *SchemaAnalyst* studied, the study concludes that *SchemaAnalyst* is $O(n^2)$ with respect to the number of check constraints in the input schema.

Keywords First keyword · Second keyword · More

1 Introduction

Search-based algorithms allow the application of guidance to problems. The algorithm attempts to improve a potential solution until the solution is acceptable. Without the use of a search-based strategy, a problem might be approached with a random sampling or greedy technique. In the domain of data generation for software testing, this means that rather than randomly

F. Author
first address
Tel.: +123-45-678910
Fax: +123-45-678910
E-mail: fauthor@example.com

S. Author
second address

selecting inputs from a program's input space, the data generator can actively seek out qualities of an input that best fulfill the test's goals McMinn (2004). While this technique has been applied to various problems, including test suite prioritization Walcott et al (2006) and testing relational database schemas Kapfhammer et al (2013), as far as we know, no research has been done on evaluating the efficiency of search-based test data generation.

This paper presents an empirical study of the search-based data generation tool, called *SchemaAnalyst*, which generates test suites for relational database schemas. To evaluate *SchemaAnalyst*, a tool was implemented in Java to systematically double the size of the program's input and record the change in it's execution time. Using this technique, for the coverage criterion and data generator tested, *SchemaAnalyst* was found to be $O(n^2)$ with respect to the number of check constraints in the schema. The contributions of this paper are therefore as follows:

1. A framework for automated doubling experiments
2. An empirical study evaluating the efficiency of a search-based data generation tool

2 Background

Worst-case time complexity is a useful measure of an algorithms efficiency, or how increasing the size of the input n increases the execution time of the algorithm, $f(n)$. This relationship is often expressed in big-Oh notation, where $f(n)$ is $O(g(n))$ means that the time increases by order of $g(n)$. The worst-case complexity of an algorithm is evident when n is large Goodrich and Tamassia (1998). One approach for determining the big-Oh complexity of an algorithm is to conduct a doubling experiment. By measuring the time needed to run the algorithm on an input n, and the time needed to run on $2n$, the order of growth of the algorithm can be determined McGeoch (2012); Sedgewick and Schidlowsky (1998).

Intuitively, the goal of a doubling experiment is to draw a conclusion regarding the efficiency of the algorithm from the ratio $f(2n)/f(n)$. This ratio represents the factor of change in runtime from input n to $2n$. A ratio of 2 would indicate that doubling the input resulted in runtime doubling. We could then conclude that the algorithm under study is $O(n)$ or $O(n \log n)$.

Ratio	Conclusion
1	Constant or Logarithmic
2	Linear or Linearithmic
4	Quadratic
8	Cubic
x	$O(n^{\log x})$

Table 1 Conclusions regarding efficiency that can be drawn from doubling ratio.

Algorithm 1 Run Doubling Experiment

```
while Not Convergent —— N not large enough do
  for trials do
    Run Test
  end for
  Double Schema
end while
```

Algorithm 2 Convergent

```
difference = $|(r_4 - r_3) + (r_3 - r_2) + (r_2 - r_1)|$
if difference < differanceTolerance then
  return Ratio is convergent
else
  return Ratio is not convergent
end if
```

3 Technique

To determine worst case complexity, an input n is doubled until the ratio $f(2n)/f(n)$ converges to a stable value. To account for random error, every time n is doubled, $f(n)$ is recorded ten times , and the median time is used for calculating the ratios. We chose median to minimize the effect of outliers. If mean is used instead, a single abnormally long run could have a large effect on the result. The overall structure of the experiment is shown in Algorithm 1.

This convergence checking is necessary because of the fact that worst-case time is only apparent for large values of n. If too few doubles are tried, then the experiment may terminate before n reaches a value where the true worst-case time complexity is apparent. At the same time, for inefficient algorithms, each additional double run incurs a substantial time overhead. To conduct the testing efficiently, the experiment should terminate as quickly as possible.

To test for convergence, the last four ratios are compared, and the sum of differences between them is compared to a tolerance value. The convergence algorithm is shown as Algorithm 2.

Another consequence of worst-case time only being apparent for large n, is that a very small initial n may appear to converge to 1, which indicates constant time. To prevent the experiment from incorrectly terminating given a small starting n, we require that a program under study display a ratio of 1 for many runs before judging that the ratio does in fact converge to 1. Because 1 signifies constant or logarithmic time, requiring these doubles does not significantly increase the time needed to run the experiment, while providing assurance that a small ratio is not due to an insufficiently small n. This rule is shown in Algorithm 3.

Algorithm 3 N Large Enough

```
if ratio $\approx 1$ then
    if number of doubles < *doublesTolerance* then
        return N is not large enough
    end if
end if
return N is large enough
```

4 Experimental Design

In our experimental study, we set *differanceTolerance* to 0.40. This value was chosen by performing doubling experiments on various algorithms with known worst case time complexities, and observing that the ratio converged to the correct value when *diff* < 0.40.

We set *doublesTolerance* to 20 after observing that *SchemaAnalyst* stopped displaying constant behavior after around 5 doubles.

To analyze *SchemaAnalyst*, the iTrust and NistWeather case studies provided by *SchemaAnalyst* were used as the initial input schemas. Both of these schemas are taken from real world applications. The factor n under study was the number of check constraints present on the schema. A Java program was implemented to double the number of these constraints. Generating synthetic check constraints is non-trivial because there are many possible check constraints, and generating a constraint that is unsatisfiable might cause the data generation tool to take a longer amount of time than should be the case. To avoid this problem, we instead duplicate the existing check constraints present on the schema rather than attempt to generate new ones. This technique is easy to implement and ensures that the check constraints added are semantically valid. For every table in the input schema, the tool duplicated the existing check constraints and added the duplicates to the table.

The test suite generation tool provided by *SchemaAnalyst* requires a coverage criterion and a data generator to be specified. A coverage criterion is a system of rules that generate test requirements Ammann and Offutt (2008). The data generator is the object that generates the test data according to the rules specified by the coverage criterion. The criterion used in the experiment was CONSTRAINTCACCOVERAGE, the data generator was DIRECTEDRANDOM. The doubling experimentation software were implemented in Java, and were both compiled and run using version 1.7 of the compiler and Java Virtual Machine. The experiment was executed on an Ubuntu 13.10 machine with a 2.4 GHz quad core CPU running the 3.11.0-18-generic x86_64 GNU/Linux kernel and 4GB of memory.

5 Results

Our technique was able to determine the worst-case time complexity of *SchemaAnalyst* with regard to the number of check constraints on the input schema,

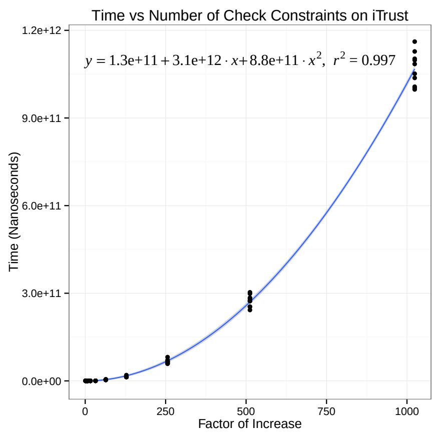

Fig. 1 Time vs Check Constraints on iTrust.

using the constraintCACCoverage coverage criterion and the directedRandom data generator. Under these conditions, *SchemaAnalyst* displayed $O(n^2)$ behavior. Figures 1 and 2 show the data collected during the experiment with a quadratic fit. The horizontal axis shows by what factor the number of initial check constraints i have been increased. For $x = 100$, the number of check constraints on the schema is $100i$. The vertical axis shows the time in nanoseconds *SchemaAnalyst* needed to generate a test suite for the input schema. Each point on the graph represents the amount of time *SchemaAnalyst* ran for a schema with ix check constraints. The equation shown is the best fit quadratic equation for that data. A quadratic relationship indicates that *SchemaAnalyst* is $O(n^2)$. The value r^2 is a measure of the quality of fit between a model and the data. The closer r^2 is to 1, the better the quality of the model. An r^2 of .997 and 1 indicates that the models are a good fit for the data.

Threats to Validity

Our technique for doubling the number of check constraints on the schema is simply to duplicate the existing check constraints. It is possible that *SchemaAnalyst* does less work processing these copied check constraints than it would given unique check constraints. However, doubling the check constraints in this way is an easy to implement, semantically significant way of evaluating *SchemaAnalyst*.

Additionally, since worst-case time is only apparent for large n, it is possible that the experiment terminated too quickly. To guard against this problem, Algorithms 2 and 3 were tested on various other algorithms with known worst-case complexities, and found to be reliable.

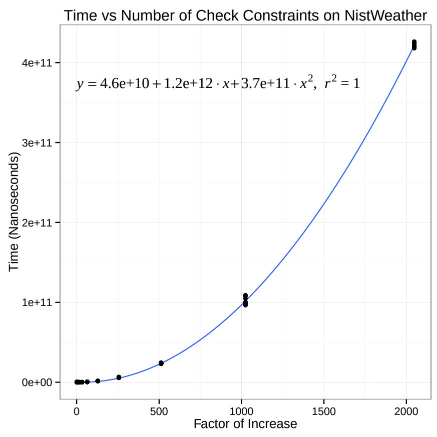

Fig. 2 Time vs Check Constraints on NistWeather.

6 Related Work

Goldsmith et al. developed a system to empirically evaluate computational complexly. Their system, *Trend-Prof*, uses code instrumentation to count the number of times each block of code is executed, and then groups these blocks by their behavior. *Trend-Prof* takes in a collection of workloads, user specified features of the workloads, and the program to be studied. This technique results in a more powerful analysis. However, the authors do not address the issue of generating the workloads necessary to achieve a meaningful result, and we attempt to do this automatically. Additionally, our approach is novel because we apply it to a domain where the theoretical scalability is not yet known. Goldsmith et al (2007)

7 Conclusions and Future Work

The automated doubling experiment was able to determine the worst case time complexity of *SchemaAnalyst* with respect to the number of check constraints in the input schema, for the constraintCACCoverage criterion and the directedRandom data generator. Additional experiments will be conducted on other criterions and data generators. Additionally, other factors that may influence the runtime of schema analysis, such as the number of primary keys, foreign keys, tables, columns, etc will be investigated.

References

Ammann P, Offutt J (2008) Introduction to software testing. Cambridge University Press

Goldsmith SF, Aiken AS, Wilkerson DS (2007) Measuring empirical computational complexity. In: Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering, ACM, New York, NY, USA, ESEC-FSE '07, pp 395–404, DOI 10.1145/1287624.1287681, URL http://doi.acm.org/10.1145/1287624.1287681

Goodrich MT, Tamassia R (1998) Data structures and algorithms in Java. World wide series in computer science, Wiley

Kapfhammer GM, McMinn P, Wright CJ (2013) Search-based testing of relational schema integrity constraints across multiple database management systems. In: International Conference on Software Testing, Verification and Validation (ICST 2013), IEEE

McGeoch CC (2012) A Guide to Experimental Algorithmics. Cambridge University Press

McMinn P (2004) Search-based software test data generation: A survey. Software Testing, Verification and Reliability 14(2):105–156, DOI 10.1002/stvr.294, URL http://philmcminn.staff.shef.ac.uk/publications/pdfs/2004-stvr.pdf

Sedgewick R, Schidlowsky M (1998) Algorithms in Java, Third Edition, Parts 1-4: Fundamentals, Data Structures, Sorting, Searching, 3rd edn. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA

Walcott KR, Soffa ML, Kapfhammer GM, Roos RS (2006) Timeaware test suite prioritization. In: Proceedings of the 2006 International Symposium on Software Testing and Analysis, ACM, New York, NY, USA, ISSTA '06, pp 1–12, DOI 10.1145/1146238.1146240, URL http://doi.acm.org/10.1145/1146238.1146240