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

Cody Kinneer
Allegheny College
kinneerc@allegheny.edu

Luke Smith
Allegheny College
smithl4@allegheny.edu

Gregory Kapfhammer
Allegheny College
gkapfham@allegheny.edu

ABSTRACT

When evaluating an algorithm, it is often useful to speak of its efficiency in terms of its worst case complexity. However, for certain cases such as search-based algorithms, determining an algorithm's efficiency by theoretical analysis is unfeasible. 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. This method is then applied to the domain of data generation for relational database schemas. A technique for systematically doubling the size of schemas was implemented, and an empirical study was conducted on the search-based data generation tool *SchemaAnalyst*. For the parameters of *SchemaAnalyst* testing, the study concluded that *SchemaAnalyst* was $O(n^2)$ with respect to the number of check constraints in the input schema.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

1. INTRODUCTION

Search-based algorithms allow guidance to be applied to problems that might otherwise be approached with a random sampling technique. In the domain of data generation for software testing, this means that rather than randomly selecting inputs from a program's input space, the qualities of the input that best fulfill the test's goals can be actively sought out by the data generator [5]. While this technique has been applied to various problems, including test suite prioritization [7] and testing relational database schemas [3], 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 *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 programs input and record the change in it's execution time. Using this technique, *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 [2]. One approach for determining the worst-case complexity of an algorithm f is to conduct a doubling experiment. By measuring the time needed to run the algorithm on n, and the time needed to run on $2n$, the order of growth of f can be determined [1, 6].

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 $O(n)$.

3. TECHNIQUE

To determine worst case complexity, an input n was doubled until the ratio $f(2n)/f(n)$ converged to a stable value. To account for random error, every time n was doubled, $f(n)$ was recorded ten times, and the median time was used for calculating the ratio. 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 tested then the experiment may terminate before n reaches a value where the worst case time is apparent. At the same time, for inefficient algorithms, each additional double tested incurs a substantial time cost, so to conduct the testing efficiently the experiment should terminate as quickly as possible.

To test for convergence, the last four ratios were compared, and the sum of differences between them is compared to a tolerance value. 0.40 was chosen by performing doubling experiments on various algorithms with known worst case

Algorithm 1 Run Doubling Experiment

```
while Diff not convergent || N not large enough do
    for count < 10 do
        Run Test
        count++
    end for
    Double Schema
end while
```

Algorithm 2 Diff not Convergent

```
diff = |(r4 − r3) + (r3 − r2) + (r2 − r1)|
if diff < 0.40 then
    return FALSE
else
    return TRUE
end if
```

Algorithm 3 N not Large Enough

```
if ratio ≈ 1 then
    if Doubles < 20 then
        minRuns++
        return TRUE
    end if
end if
return FALSE
```

time complexities, and observing that the ratio converged to the correct value when $diff < 0.40$. 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 or logarithmic time. To prevent the experiment from incorrectly terminating given a small starting n, we require that an algorithm under test display a ratio of 1 for many runs before judging that the ratio does in fact converge to 1. In this case, the experiment is considered convergent if the ratio remains 1 for twenty consecutive doubles. 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 test is shown as Algorithm 3.

4. EXPERIMENTAL DESIGN

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 tool 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 [1]. 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 factory was DIRECTEDRANDOM. The testing tools were implemented in Java, and were both compiled and run using version 1.7 of the compiler and JVM. 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.

5. RESULTS

Our technique was successfully able to determine the worst case time complexity of *SchemaAnalyst* with regard to the number of check constraints on the input schema, 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 x-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 y-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 $O(n^2)$. 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.

6. 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 factories. 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.

[***] After further thought, I am not sure that this is the best title. It does not explain what is novel about the paper. Needs to connect to — "automated" — "time complexity" — "empirical"

[**]

Again, this is a parameter to an algorithm or a tool. It should be a variable when you are talking about your method. Then, in Section 4 you can explain the values that you picked. It would also be best if you performed a sensitivity analysis to show that the results of your empirical study do not vary too much if you happen to pick different values. (Or, is that not the case?? If there is sensitivity, then we also need to do some further explaining).

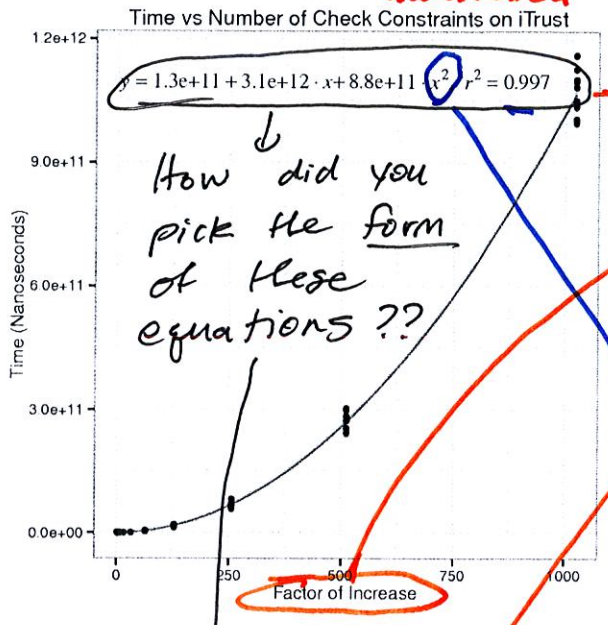

Figure 1: Time vs Check Constraints on iTrust.

You may want to connect this to some notation and then make sure that the paper clearly explains this term and gives an example.

okay — so, this also confirms that schema analyst is a $\Theta(n^3)$ algorithm.

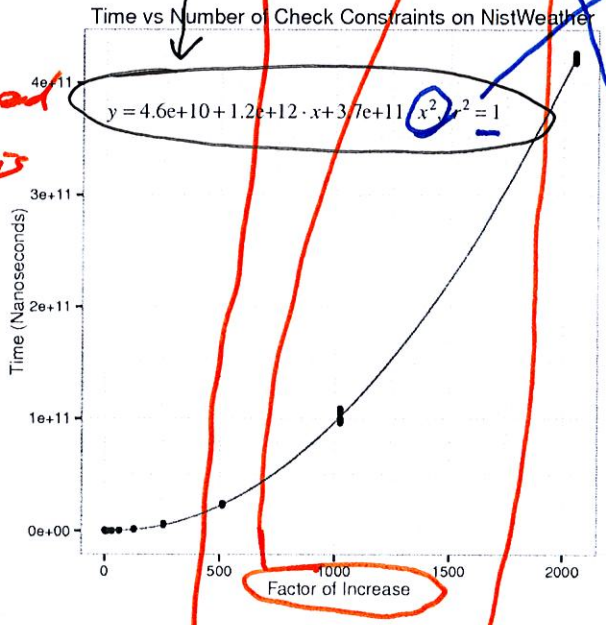

Figure 2: Time vs Check Constraints on NistWeather.

Please note that the caption is slightly different than the axis label.

[*]

The final version of this paper needs to give a full description of all of the doubling methods for relational database schemas. Whenever possible, there should be technical diagrams that show how the doubling works.

Q: What are the evaluation metrics for the experiments?

Q: What are the research questions for the experiments?

Q: ?? How do you eval. the effect of a doubler??

7. REFERENCES

[1] P. Ammann and J. Offutt. *Introduction to software testing*. Cambridge University Press, 2008.

[2] M. T. Goodrich and R. Tamassia. *Data structures and algorithms in Java*. World wide series in computer science. Wiley, 1998.

[3] G. M. Kapfhammer, P. McMinn, and C. J. Wright. 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, March 2013.

[4] C. C. McGeoch. *A Guide to Experimental Algorithmics*. Cambridge University Press, 2012.

[5] P. McMinn. *Evolutionary Search for Test Data in the Presence of State Behaviour*. PhD thesis, The University of Sheffield, 2005.

[6] R. Sedgewick and M. Schidlowsky. *Algorithms in Java, Third Edition, Parts 1-4: Fundamentals, Data Structures, Sorting, Searching*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3rd edition, 1998.

[7] K. R. Walcott, M. L. Soffa, G. M. Kapfhammer, and R. S. Roos. Timeaware test suite prioritization. In *Proceedings of the 2006 International Symposium on Software Testing and Analysis*, ISSTA '06, pages 1–12, New York, NY, USA, 2006. ACM.

hyphen needed

NOTE: The paper needs to give one or more examples of
→ A schema
→ schema Analyst & its inputs & outputs (use a diag)

This would be in a background section, I think

→ It is okay to cite Phil's PhD thesis. But, we really need to cite the JSTVR survey paper that he wrote. this is the best fit for your goal.

Also your system is a little like Daikon by Michael Ernst.

FUTURE WORK IDEA:

It would be really cool if we could implement doublers for a Java program (i.e., double the # of methods, # of params to a method) and then use this as input to an ATDG tool like EvoSuite.

Important Reminder:

Please make sure that you read the paper "Measuring Empirical computational Complexity" published by Goldsmith et al. at FSE 2007. Also, you should look at their trend-prof tool is, as I recall, freely available for download. This paper does some things similar to what we do. What are the similarities and differences? Are there any way(s) in which we are better or worse?

→ trend-prof.tigris.org

→ would it be possible to use this tool? no, not directly for our system because it seems to work only for C/C++ programs.

For the conclusion AP — say what CW + FW = (?)