Using Test Suite Prioritization to Avoid Database Restarts

Joshua J. Geiger

Joshua J. Geiger
joshua.geiger@gmail.com

Department of Computer Science
Allegheny College
http://cs.allegheny.edu

Gregory M. Kapfhammer
gkapfham@allegheny.edu

Software Testing Challenges

- Applications increasingly need to store, manage, and retrieve large amounts of data.
- Databases are frequently used for this task.
- Developers write ***unit test cases*** to ensure the correctness of their code.
- ***Regression testing*** repeatedly executes a ***test suite*** of unit tests to reveal bugs in code.
- Test cases that test a program which interacts with a database usually require the database to have an ***initial state***, or start with a specific configuration.
- Executing these test cases may alter the state of the database.

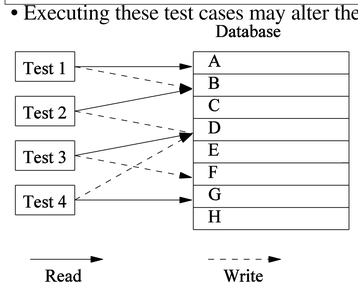

Figure 1: Test Cases Interacting with a Database: Demonstrates test cases reading and writing to a database and how they may conflict.

- Subsequent test cases operating on this altered data may report incorrect results.
- To prevent incorrect results, the database state is restored by restarting the database.
- Restarting the database between the execution of every test case ensures each test case only operates on the correct state of the database.
- Restarting the database many times throughout the execution of a large test suite is very time intensive, thus lengthening the development process and increasing costs.

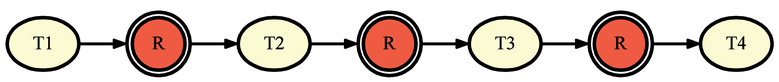

Figure 2: Naïve Test Suite Schedule: This is an example is based on the test suite in Figure 1. Restarts are scheduled between each test case to ensure test case independence.

Conflict Graphs

- Test cases read and write from different portions of the database, thus test cases do not necessarily corrupt the database for every other test case.
- A ***conflict graph*** is a digraph representation of how the test cases ***conflict*** or ***contaminate*** one another.

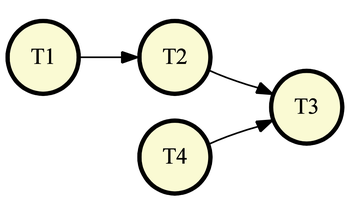

Figure 3: Conflict Graph: Represents how the test cases from Figure 1 contaminate each other.

Avoiding Restarts

Prioritization

- The test cases can be rearranged, or ***prioritized***, so that test cases that do not conflict are executed before those that do.
- When the conflict graph is acyclic, a reversed ***topological sort*** will yield a schedule without any restarts.

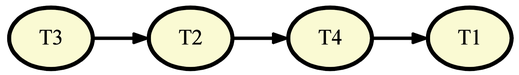

Figure 4: Prioritized Test Suite Schedule: This is an example based on Figure 1. Rearranging test cases avoids restarting the database.

Breaking Cycles

- Cycles in the graph prevent topological sorting, thus ensuring at least one restart.
- By removing test cases that create cycles, a topological sort can be performed on the remaining test cases.
- The cycles in the selected test cases are recursively removed.
- A database restart is placed into the schedule whenever a recursion occurs.

Synthetic Test Suites

- The technique was implemented and tested using synthetic conflict data.
- Conflict graphs were generated rather than created from real world test suites.
- Demonstrates the feasibility and effectiveness of this approach.

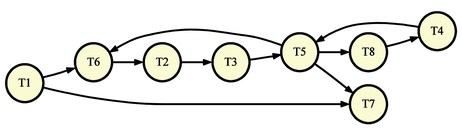

Figure 5: Conflict Graph with Cycles.

Figure 7: Cycle Breaking Technique.

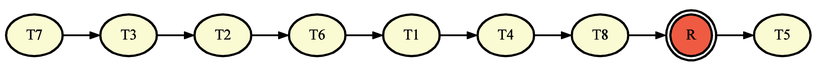

Figure 6: Prioritized Test Suite Schedule. This is a prioritized schedule of the test suite in Figure 5 created using the cycle breaking technique.

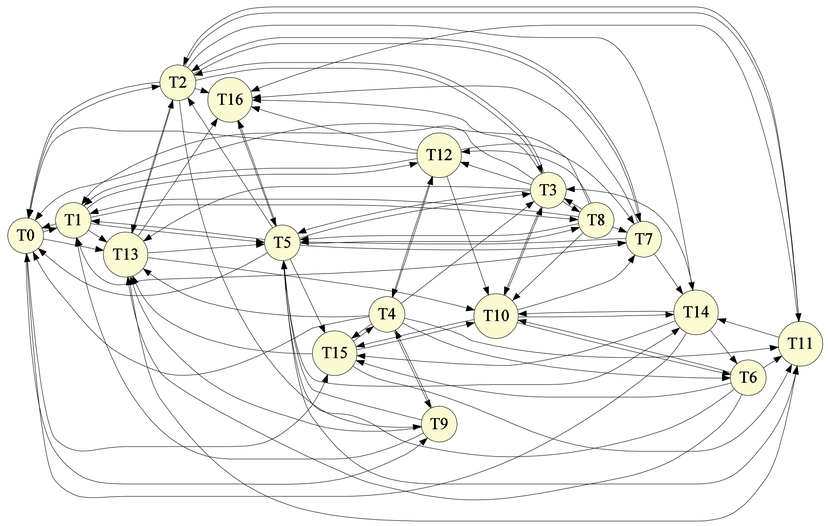

Figure 8: Sample Conflict Graph from the Synthetic Data.

Discussion

- Currently, the cycle breaker removes test cases based on which ones participate in the greatest number of cycles.
- Figure 8 contains 17 test cases, 94 conflicts, and 1,764,008 cycles. The cycle breaking technique eliminates all but two restarts.
- The number of cycles in a graph grows **exponentially** relative to the number of edges and vertices. For example, a graph with 15 test cases and 76 conflicts based on the graph in Figure 8 has only 191,379 cycles.
- The cost of breaking cycles using the current criteria is **exponential** in the number of cycles in the graph. Cycle breaking in the 15 test case conflict graph required roughly 40 seconds. In contrast, breaking cycles in the 17 test case conflict graph required over 25 minutes.

Future Work

- Break cycles by other metrics, including the cost and coverage of the test cases.
- Implement alternative prioritization techniques using custom heuristics.
- Empirically evaluate these techniques against search based methods.
- Enhance tool implementation and conduct experiments that collect conflict data from real world database-centric applications and test suites.
- Release tool and source code.
- Publish simulation study tools for educational use in studying directed cyclic graphs.