Research problems on software regression testing

{1} (*Input-dependent testing*) Test-case prioritization for image-processing software where test-cases are images (any format). Our goal is to normalize it to some representation by performing image analysis and perform code-coverage independent prioritization. No code instrumentation will be involved. Image comparison operator should be novel.

{2} Applications of Bloom filter, BDDs, and Approximation to regression testing.

{3} Light-weight static and dynamic instrumentation depending upon where change has occurred.

{4} Language-specific optimizations to regression testing.

{5} Test-suite reduction framework.

{6} Fault localization framework.

{7} Dynamic test-case prioritization.

{8} Lightweight code (IR) differencing tool.

{9} Granularity of analysis: basic-block, variable, statement, file, class, methods (functions), modules.

{10} B_c: tc1, tc2 (critical basic block)
B_g: t1, t2, t5, … (general basic block)
{B_c → B_g}: B_c's test-cases followed by B_g's test-cases (prioritized).
B_c: tf-idf ordering (within B_c's suite).
B_g: confidence based (do not run all test-cases).
Note: (i) we do not have fault information.
(ii) Statically we do not know which test-case might fail.

{11} "Interdependence of basic-blocks in CFG": Can we use this information for test-selection, prioritization, and reduction.

{12} Regression testing of logic programs such as Prolog.

{13} Regression testing of approximate programs.

{14} Regression testing of quantum programs such as Q# from Microsoft.

{15} RTS for CUDA programs: Multi-GPUs for parallel testing of single-GPU programs. What issues might come up?

{16} Automatic versus Manual synchronization for testing in parallel in the presence of shared or global variables.

{17} Parallelization of sequential program-under-test. Are there auto-parallelizers that transform sequential code into parallel.

{18} (*Symbolic execution*) In case of prioritization, there seems to be (as of now) no work leveraging symbolic execution. However, the time complexity of symbolic execution is the only problem.

Constraint: New version cannot be executed.

Allowed: Static + Dynamic analysis on older version. Only static analysis on new version. How can this be done to determine what new basic-blocks are touched by a changed (affected) path? Can we construct a new test-case prioritization based on this? In other words, how to prioritize (using symbolic analysis / execution) the execution of selected tests such that it is effective?