NodeSRT: A Selective Regression Testing Tool for Node.js Application

Yufeng Chen
University of British Columbia
yufengcy@student.ubc.ca

Motivation

- As software systems mature, the cost of running their entire regression test suite can become significant.
- Selective Regression Testing (SRT) is a technique that executes only a subset of tests the regression test suite can detect software failures more efficiently.
- Previous SRT studies mainly focused on standard desktop applications. Node.js applications are considered hard to perform test reduction because of Node's asynchronous, event-driven programming model and JavaScript is a dynamic pl.
- We present **NodeSRT**, a Selective Regression Testing framework for Node.js applications.

Selective Regression Testing

Two Phases in Selective Regression Test
- Test Selection phase: select tests based on dependency graph and changes
- Test running phase: execute selected tests

Test Selection Granularity
- File-level: builds a relationship between tests and files in the system and selects tests that reflect changed files.
- Method-level: builds a relationship between tests and methods.

SRT technique metrics
- **Inclusiveness**: the extend to which SRT technique chooses tests that are affected by the change
- **Precision**: the ability that the SRT technique omits tests that are not affected by the change
- **Efficiency:** the time and space complexity
- **Generality:** ability to function in a comprehensive and practical range of situation

* We say a selection technique is safe if it achieves 100% inclusiveness

Jest OnlyChange

- Current industry-standard SRT technique for Node.js application
- Test selection at File-level
- Static file dependency analysis with jest-haste-map module
- Fast, light-weighted, but not precise enough

Approach Overview

Our intuition for reducing the total running time is to improve the granularity of the selection technique to improve precision so that fewer tests are required to run when the regression suite is executed.

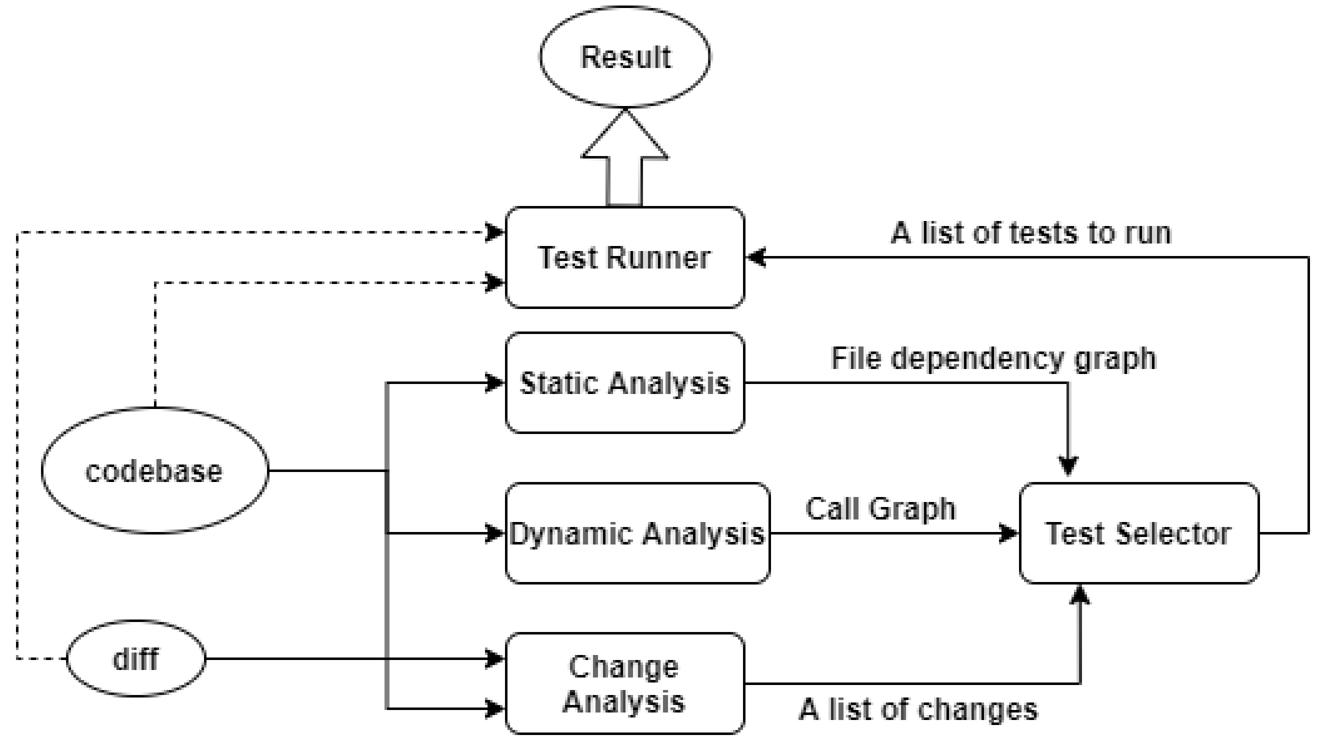
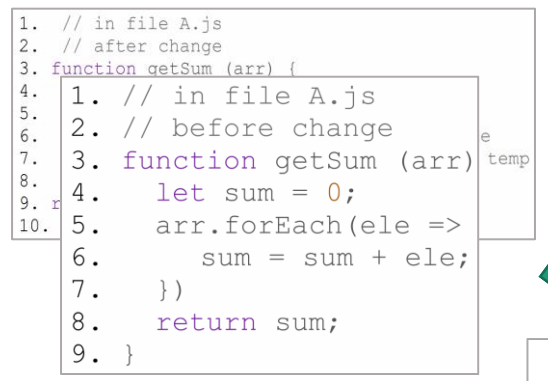


Fig. 1. NodeSRT Architecture

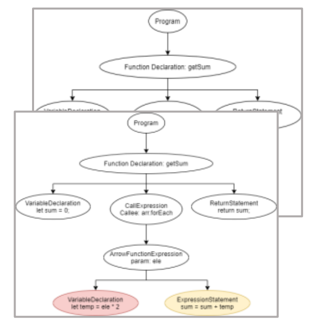
Change Analysis Module



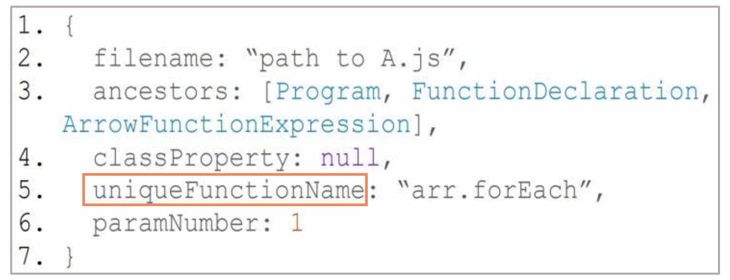
```
// in file A.js
// after change
function getSum (arr) {
```

```
// in file A.js
// before change
function getSum (arr)
  let sum = 0;
  arr.forEach(ele =>
    sum = sum + ele;
  })
  return sum;
}
```

Changed file + diff


Program
Function Declaration: getSum
VariableDeclaration let sum = 0;
CallExpression Callee: arr.forEach
ReturnStatement return sum;
ArrowFunctionExpression param: ele
VariableDeclaration let temp = ele * 2
ExpressionStatement sum = sum + temp

AST before after change



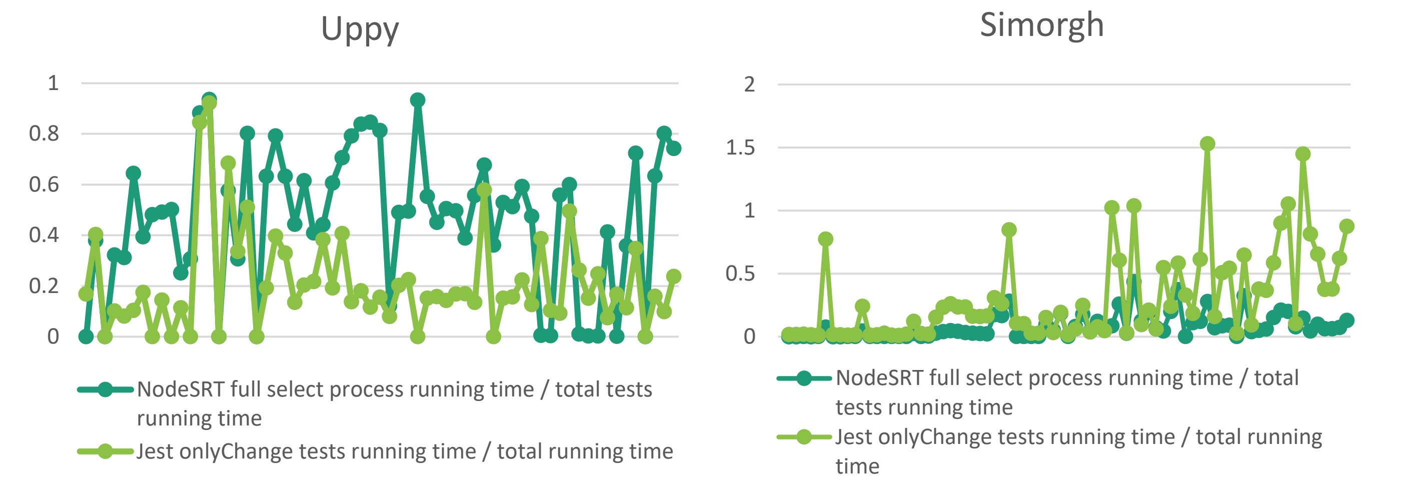
```
{
  filename: "path to A.js",
  ancestors: [Program, FunctionDeclaration,
ArrowFunctionExpression],
  classProperty: null,
  uniqueFunctionName: "arr.forEach",
  paramNumber: 1
}
```

Changed functions and their ancestors

Generate a "Unique Function Name" if it's anonymous function

Empirical Evaluation

- Experiment subject: Uppy (112k lines of code, 216 unit tests, 20% of code coverage) and Simorgh (698k lines of code, 2801 unit tests, 97% of code coverage)
- We performed test selection on a total of 588 commits from the two subjects. For each commit, we generate a diff patch from the previous commit to serving as input to NodeSRT.



Comparison on Selected Test running time

Project Name	Total Round	Retest all time (s)	Avg. Selection time (ms)	Avg. NodeSRT selected test number	Avg. Jest selected test number	Avg. NodeSRT total running time	Avg. Jest total running time
Uppy	480	69.3	490	8.4 %	20.7 %	50.2 %	23.6 %
Simorgh	108	450.7	2256	2.7 %	17.2 %	8.1 %	30.2 %

Empirical Study Result

- NodeSRT selection step for both projects is less than 5% of total running time
- Comparing to Jest, NodeSRT selects 1.5 times fewer tests in Uppy, 5.3 times fewer tests in Simorgh
- Although NodeSRT selects fewer tests in Uppy, Jest OnlyChange runs faster than NodeSRT. This is because Jest OnlyChange makes use of Jest's own jest-haste map module and customized file system module: watchman. Future works can be done for NodeSRT in this part. For project with high code coverage: Simorgh, NodeSRT selected fewer tests and runs 2.7 times faster

Links:

- Preprint: https://arxiv.org/abs/2104.00142
- NodeSRT: https://github.com/charlie-cyf/nodeSRT
- Uppy: https://github.com/transloadit/uppy
- Simorgh: https://github.com/bbc/simorgh

pre-print