



University of
Sheffield



COM3529 Software Testing and Analysis

Regression Testing

Dr José Miguel Rojas

Roadmap

Beizer's Maturity Model

Test Automation

Unit Testing

Control/Data-Flow Analysis

Code Coverage

Mutation Testing

Regression Testing

Fuzzing

Search-based Test Generation

Model-Based Testing

Roadmap

Beizer's Maturity Model

Test Automation

Unit Testing

Control/Data-Flow Analysis

Code Coverage

Mutation Testing

Regression Testing

Fuzzing

Search-based Test Generation

Model-Based Testing

Motivation

Changes have been made to the system's code
...or to the system's environment

Have new faults been introduced that weren't there before?

Can we **confidently** deploy the new version of the system?

Regression Testing is now standard in the software development industry

Typically based on a **regression test suite**

Scenarios



Maven™

Regression testing can be **very costly**

As software grows, so do test suites

CI/CD (Continuous Integration & Development) frameworks help

Running regression test suites overnight is common practice

Rerunning existing tests **may not always be possible**, e.g. if public interfaces have changed

For embedded systems, e.g. car controllers, retesting the software **in use** may take **months**. Simulation helps but is not a complete solution.

Regression Testing Techniques

Minimisation – Could my test suite be smaller?

Typically based on a criterion, i.e., minimise while retaining ...

Prioritisation – In which order should I run my tests?

To detect new defects as soon as possible

Selection – Which tests should I run?

Given a set of lines of code

Minimisation

Minimisation

As software grows, so does the size of its test suite!

Tests become **obsolete**, or **redundant**

Test maintainability is rarely a priority in practice

Removing redundancy

Many similar tests covering the same code!

Coverage information from CI/CD, i.e., tests have been run before

Keep only tests that execute changed (or deleted) code; deem other tests redundant wrt changes, so no need to re-execute them!

Minimisation Algorithm

Focus on code coverage, e.g., **branch coverage**

Alternative approaches exist, e.g., to preserve other properties

Given a regression test suite T , **find the smallest subset $T' \in T$ such that T' and T have the same coverage**

The hope is that preserving coverage leads to us preserving effectiveness, i.e., T' is as effective as T

Not necessarily **the** smallest test suite: If the cost of test execution varies then we may want a **cheapest-to-execute** test suite

Formalisation

Given a set of coverage goals $C = \{c_1, \dots, c_k\}$ (e.g., lines)

a.k.a. **test requirements**

And a test suite $T = \{t_1, \dots, t_n\}$ that meets/covers all of them

Where each $t_i \in T$ covers a given set C_{t_i} of coverage goals

Find the smallest $T' \subseteq T$ such that
$$\bigcup_{t_i \in T'} C_{t_i} = C$$

This corresponds to the **NP-complete Set Cover Problem**

Exact Solutions vs Heuristics

Test suites are usually large – otherwise no need to minimise them!

So, our problem instances are usually large, too

Typically infeasible to solve the problem exactly: we fall back upon heuristics: **How to find a good (small enough) test suite?**

A Simple Greedy Algorithm

Start with the empty set \emptyset

Add a test t_i that covers **most goals** and remove it from the test pool

Iterations: add the next test that covers the most goals

Terminate when the test suite provides **full coverage**

A Simple Greedy Algorithm

Test\branch	b ₁	b ₂	b ₃	b ₄	b ₅	b ₆	b ₇	b ₈
A	x	x	x			x	x	x
B	x	x	x				x	x
C	x	x	x	x				
D					x	x	x	x
E	x	x	x			x	x	x

A Simple Greedy Algorithm

Starts with **A** and **E** (either order)

Test\branch	b ₁	b ₂	b ₃	b ₄	b ₅	b ₆	b ₇	b ₈
A	x	x	x			x	x	x
B	x	x	x				x	x
C	x	x	x	x				
D					x	x	x	x
E	x	x	x			x	x	x

A Simple Greedy Algorithm

Test\branch	b ₁	b ₂	b ₃	b ₄	b ₅	b ₆	b ₇	b ₈
A	x	x	x			x	x	x
B	x	x	x				x	x
C	x	x	x	x				
D					x	x	x	x
E	x	x	x			x	x	x

Starts with **A** and **E** (either order)

Next, **B** is the obvious choice

A Simple Greedy Algorithm

Test\branch	b ₁	b ₂	b ₃	b ₄	b ₅	b ₆	b ₇	b ₈
A	x	x	x			x	x	x
B	x	x	x				x	x
C	x	x	x	x				
D					x	x	x	x
E	x	x	x			x	x	x

Starts with **A** and **E** (either order)

Next, **B** is the obvious choice

Finally, **C** and **D** (either order)

A Simple Greedy Algorithm

Test\branch	b ₁	b ₂	b ₃	b ₄	b ₅	b ₆	b ₇	b ₈
A	x	x	x			x	x	x
B	x	x	x				x	x
C	x	x	x	x				
D					x	x	x	x
E	x	x	x			x	x	x

Starts with **A** and **E** (either order)

Next, **B** is the obvious choice

Finally, **C** and **D** (either order)

Resulting set: **{A, E, B, C, D}**

A Simple Greedy Algorithm

Test\branch	b ₁	b ₂	b ₃	b ₄	b ₅	b ₆	b ₇	b ₈
A	x	x	x			x	x	x
B	x	x	x				x	x
C	x	x	x	x				
D					x	x	x	x
E	x	x	x			x	x	x

Starts with **A** and **E** (either order)

Next, **B** is the obvious choice

Finally, **C** and **D** (either order)

Resulting set: **{A, E, B, C, D}**

But could have just used {C, D}!

A Simple Greedy Algorithm

Test\branch	b ₁	b ₂	b ₃	b ₄	b ₅	b ₆	b ₇	b ₈
A	x	x	x			x	x	x
B	x	x	x				x	x
C	x	x	x	x				
D					x	x	x	x
E	x	x	x			x	x	x

Starts with **A** and **E** (either order)

Next, **B** is the obvious choice

Finally, **C** and **D** (either order)

Resulting set: **{A, E, B, C, D}**

But could have just used {C, D}!

Not always a good approach!

Additional Greedy

Take into account the coverage **already achieved!**

Start again with the empty set \emptyset

Add one of the tests that covers most goals

Iterations: add a test that covers most **currently uncovered** goals

Terminate with **full coverage**

Effective wrt Set Cover Problem – good approximation

Additional Greedy

Test\branch	b ₁	b ₂	b ₃	b ₄	b ₅	b ₆	b ₇	b ₈
A	x	x	x			x	x	x
B	x	x	x				x	x
C	x	x	x	x				
D					x	x	x	x
E	x	x	x			x	x	x

Additional Greedy

Starts with **A** or **E** (either one)

Test\branch	b ₁	b ₂	b ₃	b ₄	b ₅	b ₆	b ₇	b ₈
A	x	x	x			x	x	x
B	x	x	x				x	x
C	x	x	x	x				
D					x	x	x	x
E	x	x	x			x	x	x

Additional Greedy

Starts with **A** or **E** (either one)

Uncovered goals: **b₄** and **b₅**

Test\branch	b ₁	b ₂	b ₃	b ₄	b ₅	b ₆	b ₇	b ₈
A	x	x	x			x	x	x
B	x	x	x				x	x
C	x	x	x	x				
D					x	x	x	x
E	x	x	x			x	x	x

Additional Greedy

Test\branch	b ₁	b ₂	b ₃	b ₄	b ₅	b ₆	b ₇	b ₈
A	x	x	x			x	x	x
B	x	x	x				x	x
C	x	x	x	x				
D					x	x	x	x
E	x	x	x			x	x	x

Starts with **A** or **E** (either one)

Uncovered goals: **b₄** and **b₅**

Add **C** and **D** (either order)

Additional Greedy

Test\branch	b ₁	b ₂	b ₃	b ₄	b ₅	b ₆	b ₇	b ₈
A	x	x	x			x	x	x
B	x	x	x				x	x
C	x	x	x	x				
D					x	x	x	x
E	x	x	x			x	x	x

Starts with **A** or **E** (either one)

Uncovered goals: **b₄** and **b₅**

Add **C** and **D** (either order)

Each covers one uncovered goal

Additional Greedy

Test\branch	b ₁	b ₂	b ₃	b ₄	b ₅	b ₆	b ₇	b ₈
A	x	x	x			x	x	x
B	x	x	x				x	x
C	x	x	x	x				
D					x	x	x	x
E	x	x	x			x	x	x

Starts with **A** or **E** (either one)

Uncovered goals: **b₄** and **b₅**

Add **C** and **D** (either order)

Each covers one uncovered goal

Resulting set: **{A, C, D}**

Additional Greedy

Test\branch	b ₁	b ₂	b ₃	b ₄	b ₅	b ₆	b ₇	b ₈
A	x	x	x			x	x	x
B	x	x	x				x	x
C	x	x	x	x				
D					x	x	x	x
E	x	x	x			x	x	x

Starts with **A** or **E** (either one)

Uncovered goals: **b₄** and **b₅**

Add **C** and **D** (either order)

Each covers one uncovered goal

Resulting set: **{A, C, D}**

Not optimal, but better!

Harrold et al (1993)

Test\branch	b ₁	b ₂	b ₃	b ₄	b ₅	b ₆	b ₇	b ₈
A	x	x	x			x	x	x
B	x	x	x				x	x
C	x	x	x	x				
D					x	x	x	x
E	x	x	x			x	x	x

Harrold et al (1993)

Idea: **Uniquely-covered** goals first

Test\branch	b ₁	b ₂	b ₃	b ₄	b ₅	b ₆	b ₇	b ₈
A	x	x	x			x	x	x
B	x	x	x				x	x
C	x	x	x	x				
D					x	x	x	x
E	x	x	x			x	x	x

Harrold et al (1993)

Idea: **Uniquely-covered** goals first
b₄ uniquely-covered by **C**; so add **C**

Test\branch	b ₁	b ₂	b ₃	b ₄	b ₅	b ₆	b ₇	b ₈
A	x	x	x			x	x	x
B	x	x	x				x	x
C	x	x	x	x				
D					x	x	x	x
E	x	x	x			x	x	x

Harrold et al (1993)

Idea: **Uniquely-covered** goals first

b₄ uniquely-covered by **C**; so add **C**

b₅ uniquely-covered by **D**; so add **D**

Test\branch	b ₁	b ₂	b ₃	b ₄	b ₅	b ₆	b ₇	b ₈
A	x	x	x			x	x	x
B	x	x	x				x	x
C	x	x	x	x				
D					x	x	x	x
E	x	x	x			x	x	x

Harrold et al (1993)

Test\branch	b ₁	b ₂	b ₃	b ₄	b ₅	b ₆	b ₇	b ₈
A	x	x	x			x	x	x
B	x	x	x				x	x
C	x	x	x	x				
D					x	x	x	x
E	x	x	x			x	x	x

Idea: **Uniquely-covered** goals first

b₄ uniquely-covered by **C**; so add **C**

b₅ uniquely-covered by **D**; so add **D**

Iterations: consider goals covered by two, three, etc tests

Harrold et al (1993)

Test\branch	b ₁	b ₂	b ₃	b ₄	b ₅	b ₆	b ₇	b ₈
A	x	x	x			x	x	x
B	x	x	x				x	x
C	x	x	x	x				
D					x	x	x	x
E	x	x	x			x	x	x

Idea: **Uniquely-covered** goals first

b₄ uniquely-covered by **C**; so add **C**

b₅ uniquely-covered by **D**; so add **D**

Iterations: consider goals covered by two, three, etc tests

Ties need resolution

Harrold et al (1993)

Test\branch	b ₁	b ₂	b ₃	b ₄	b ₅	b ₆	b ₇	b ₈
A	x	x	x			x	x	x
B	x	x	x				x	x
C	x	x	x	x				
D					x	x	x	x
E	x	x	x			x	x	x

Idea: **Uniquely-covered** goals first

b₄ uniquely-covered by **C**; so add **C**

b₅ uniquely-covered by **D**; so add **D**

Iterations: consider goals covered by two, three, etc tests

Ties need resolution

Terminate with **full coverage**

Harrold et al (1993)

Test\branch	b ₁	b ₂	b ₃	b ₄	b ₅	b ₆	b ₇	b ₈
A	x	x	x			x	x	x
B	x	x	x				x	x
C	x	x	x	x				
D					x	x	x	x
E	x	x	x			x	x	x

Idea: **Uniquely-covered** goals first

b₄ uniquely-covered by **C**; so add **C**

b₅ uniquely-covered by **D**; so add **D**

Iterations: consider goals covered by two, three, etc tests

Ties need resolution

Terminate with **full coverage**

Resulting set: **{C, D}**

Harrold et al (1993)

Test\branch	b ₁	b ₂	b ₃	b ₄	b ₅	b ₆	b ₇	b ₈
A	x	x	x			x	x	x
B	x	x	x		x		x	x
C	-	-	-	x				
D					x	x	x	x
E	x	x	x			x	x	x

Harrold et al (1993)

b₄ uniquely-covered by **C**; so **add C**

Test\branch	b ₁	b ₂	b ₃	b ₄	b ₅	b ₆	b ₇	b ₈
A	x	x	x			x	x	x
B	x	x	x		x		x	x
C	-	-	-	x				
D					x	x	x	x
E	x	x	x			x	x	x

Harrold et al (1993)

b₄ uniquely-covered by **C**; so **add C**

Next, **b₅** is covered by **2** tests: **B, D**

Test\branch	b ₁	b ₂	b ₃	b ₄	b ₅	b ₆	b ₇	b ₈
A	x	x	x			x	x	x
B	x	x	x		x		x	x
C	-	-	-	x				
D					x	x	x	x
E	x	x	x			x	x	x

Harrold et al (1993)

Test\branch	b ₁	b ₂	b ₃	b ₄	b ₅	b ₆	b ₇	b ₈
A	x	x	x			x	x	x
B	x	x	x		x		x	x
C	-	-	-	x				
D					x	x	x	x
E	x	x	x			x	x	x

b₄ uniquely-covered by **C**; so **add C**

Next, **b₅** is covered by **2** tests: **B, D**

Tie resolution between **B** and **D**:

Harrold et al (1993)

Test\branch	b ₁	b ₂	b ₃	b ₄	b ₅	b ₆	b ₇	b ₈
A	x	x	x			x	x	x
B	x	x	x		x		x	x
C	-	-	-	x				
D					x	x	x	x
E	x	x	x			x	x	x

b₄ uniquely-covered by **C**; so **add C**

Next, **b₅** is covered by **2** tests: **B**, **D**

Tie resolution between **B** and **D**:

Look at goals covered by **3** tests

Harrold et al (1993)

Test\branch	b ₁	b ₂	b ₃	b ₄	b ₅	b ₆	b ₇	b ₈
A	x	x	x			x	x	x
B	x	x	x		x		x	x
C	-	-	-	x				
D					x	x	x	x
E	x	x	x			x	x	x

b₄ uniquely-covered by **C**; so **add C**

Next, **b₅** is covered by **2** tests: **B, D**

Tie resolution between **B** and **D**:

Look at goals covered by **3** tests

B : b₁, b₂, b₃ vs **D : b₆** ; so **add B**

Harrold et al (1993)

Test\branch	b ₁	b ₂	b ₃	b ₄	b ₅	b ₆	b ₇	b ₈
A	x	x	x			x	x	x
B	x	x	x		x		x	x
C	-	-	-	x				
D					x	x	x	x
E	x	x	x			x	x	x

b₄ uniquely-covered by **C**; so **add C**

Next, **b₅** is covered by **2** tests: **B, D**

Tie resolution between **B** and **D**:

Look at goals covered by **3** tests

B : b₁, b₂, b₃ vs **D : b₆** ; so **add B**

Next, **b₆** is covered by **3** tests: **A, D, E**

Harrold et al (1993)

Test\branch	b ₁	b ₂	b ₃	b ₄	b ₅	b ₆	b ₇	b ₈
A	x	x	x			x	x	x
B	x	x	x		x		x	x
C	-	-	-	x				
D					x	x	x	x
E	x	x	x			x	x	x

b₄ uniquely-covered by **C**; so **add C**

Next, **b₅** is covered by **2** tests: **B, D**

Tie resolution between **B** and **D**:

Look at goals covered by **3** tests

B : b₁, b₂, b₃ vs **D : b₆** ; so **add B**

Next, **b₆** is covered by **3** tests: **A, D, E**

Resulting set: **{C, B, A|D|E}**

Multi-Objective Approach

Metaheuristics, e.g., Genetic Algorithms

Enough improvements over Greedy to be worthwhile?

Problem generalisation: Find a regression test suite T' such that **there is no smaller test suite** that provides the same coverage as T' .

Optimise two objective functions: **maximise** coverage & **minimise** cost

More flexible than Greedy: other coverage criteria as added targets

Multi-objective optimisation algorithms return a set of solutions with trade-offs

Multi-Objective Approach

Pareto Dominance: Approach to comparing candidate solutions

Given two candidate solutions \mathbf{x} and \mathbf{y} , \mathbf{x} Pareto-dominates \mathbf{y} if \mathbf{x} is **at least as good as \mathbf{y}** on all objectives and **strictly better** than \mathbf{y} on at least one objective

i.e., we would **never** choose \mathbf{y} over \mathbf{x}

Ideally, we want to find the **Pareto Front**: the set of solutions that are not Pareto-dominated by any other solution

Many metaheuristic algorithms, the most famous is probably the Non-dominated Sorting Genetic Algorithm II (NSGA-II); there is now NSGA-III

Prioritisation

Prioritisation

Goal: Find a good **test execution order**

To meet test requirements, e.g., coverage, **as early as possible**

Ideally, we would like any test failures to occur **as early as possible**

Speed up software development: **Stop testing once a test fails**

Even if we plan to execute all tests, the sooner we find failures the sooner we can start trying to fix the code

Problem

Problem: We do not know in advance **which tests will fail!**

So, the best order is **unknown**

Idea: use metrics and historical information associated with faults

Prioritise tests that are deemed **more likely** to lead to failures

While quickly **maximising coverage**

Hope: **Find faults early!**

Using Coverage

We might only look at coverage

- Aim to achieve 100% coverage as quickly as possible

- Maximise coverage for a given budget, e.g. number of tests

- Achieve coverage **quickly** to get **good** coverage whenever we stop

Greedy Algorithm

Start with **A**, then choose **D**, then **C**

To achieve 100% coverage **faster**,
we should do **C** then **D** or viceversa

Suboptimal if we must stop after just
one test

Test\branch	b ₁	b ₂	b ₃	b ₄	b ₅	b ₆	b ₇	b ₈
A	x	x	x			x	x	x
B	x	x	x				x	x
C	x	x	x	x				
D					x	x	x	x

Using Fault-Detection Data

Using Fault-Detection Data

Use Mutation Testing data!

Using Fault-Detection Data

Use Mutation Testing data!

15 mutants in total

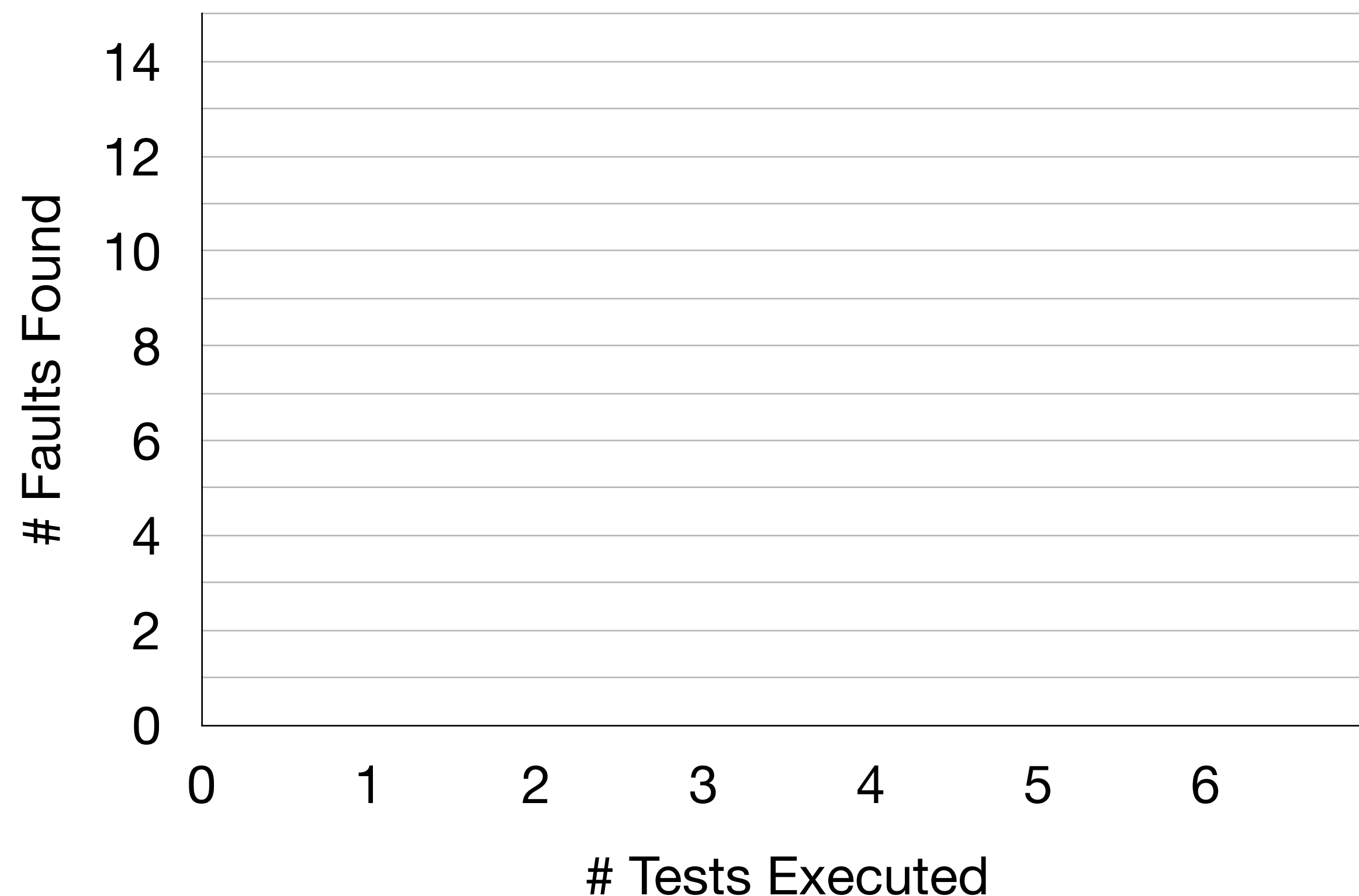
Using Fault-Detection Data

Use Mutation Testing data!

15 mutants in total

Tests A-F kill 13 of them

Using Fault-Detection Data



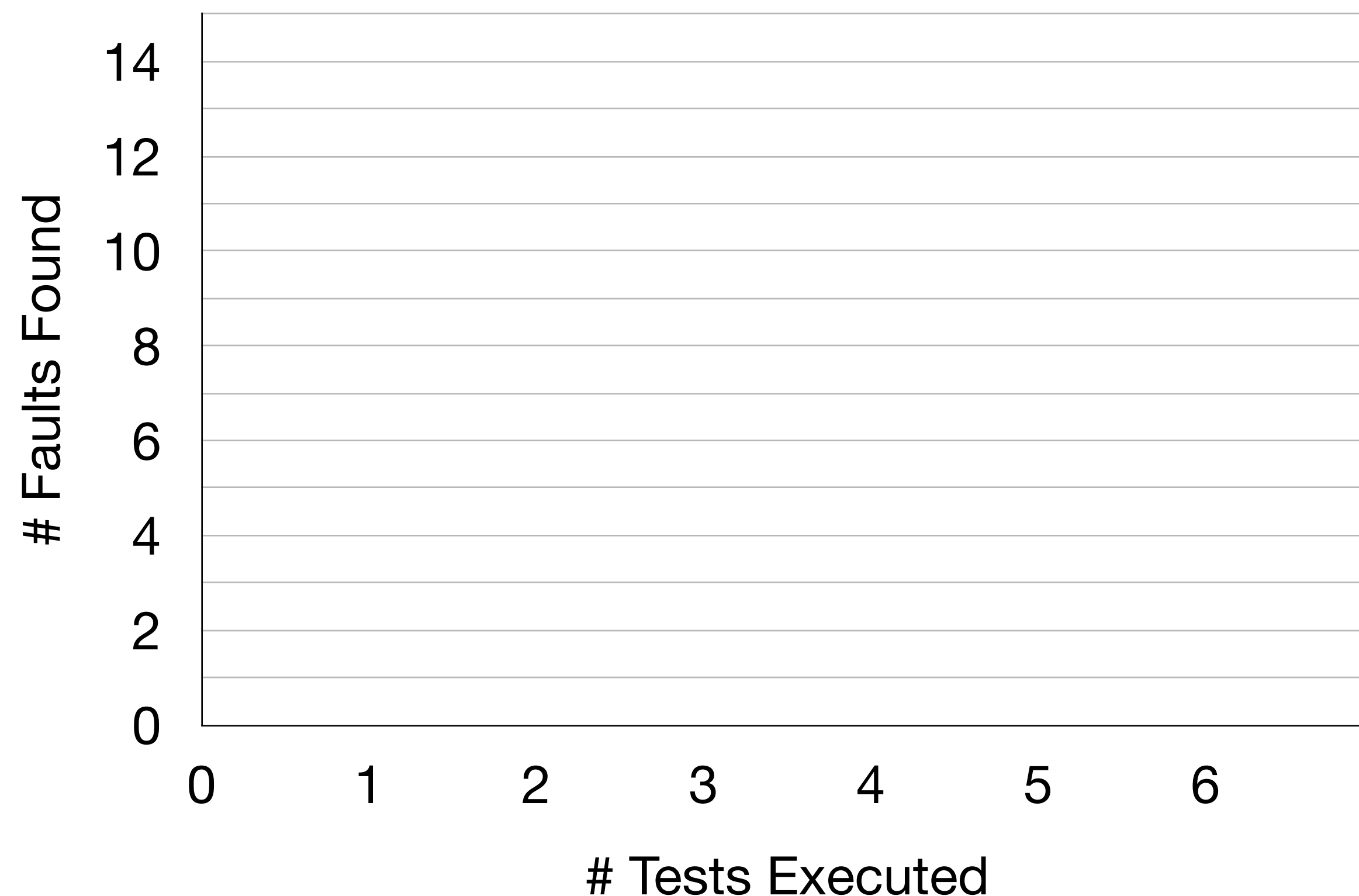
Use Mutation Testing data!

15 mutants in total

Tests A-F kill 13 of them

- Random
- Total Coverage
- Adding Coverage

Using Fault-Detection Data



- Random
- Total Coverage
- Adding Coverage

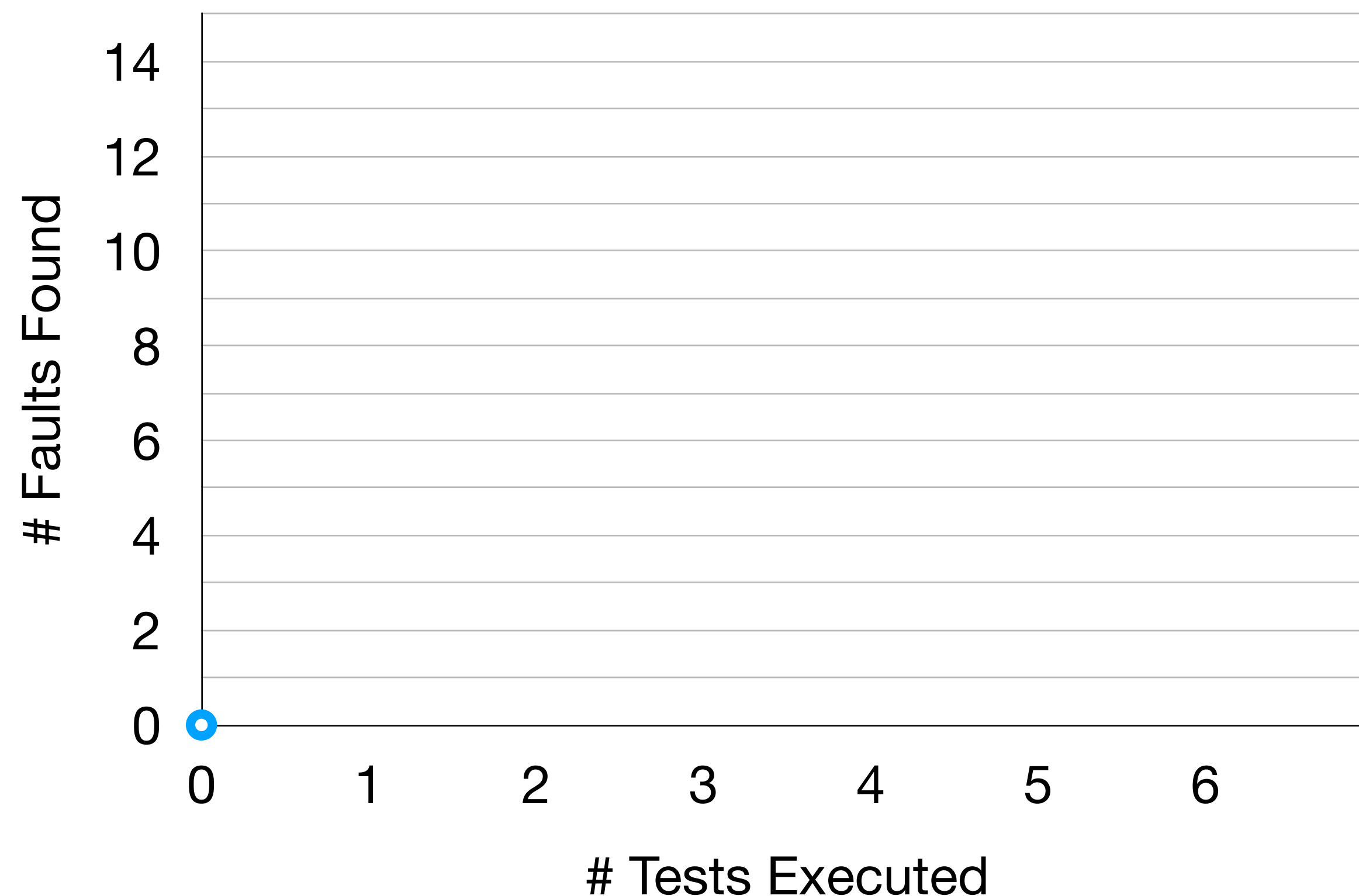
Use Mutation Testing data!

15 mutants in total

Tests A-F kill 13 of them

Random: E-A-B-F-D-C

Using Fault-Detection Data



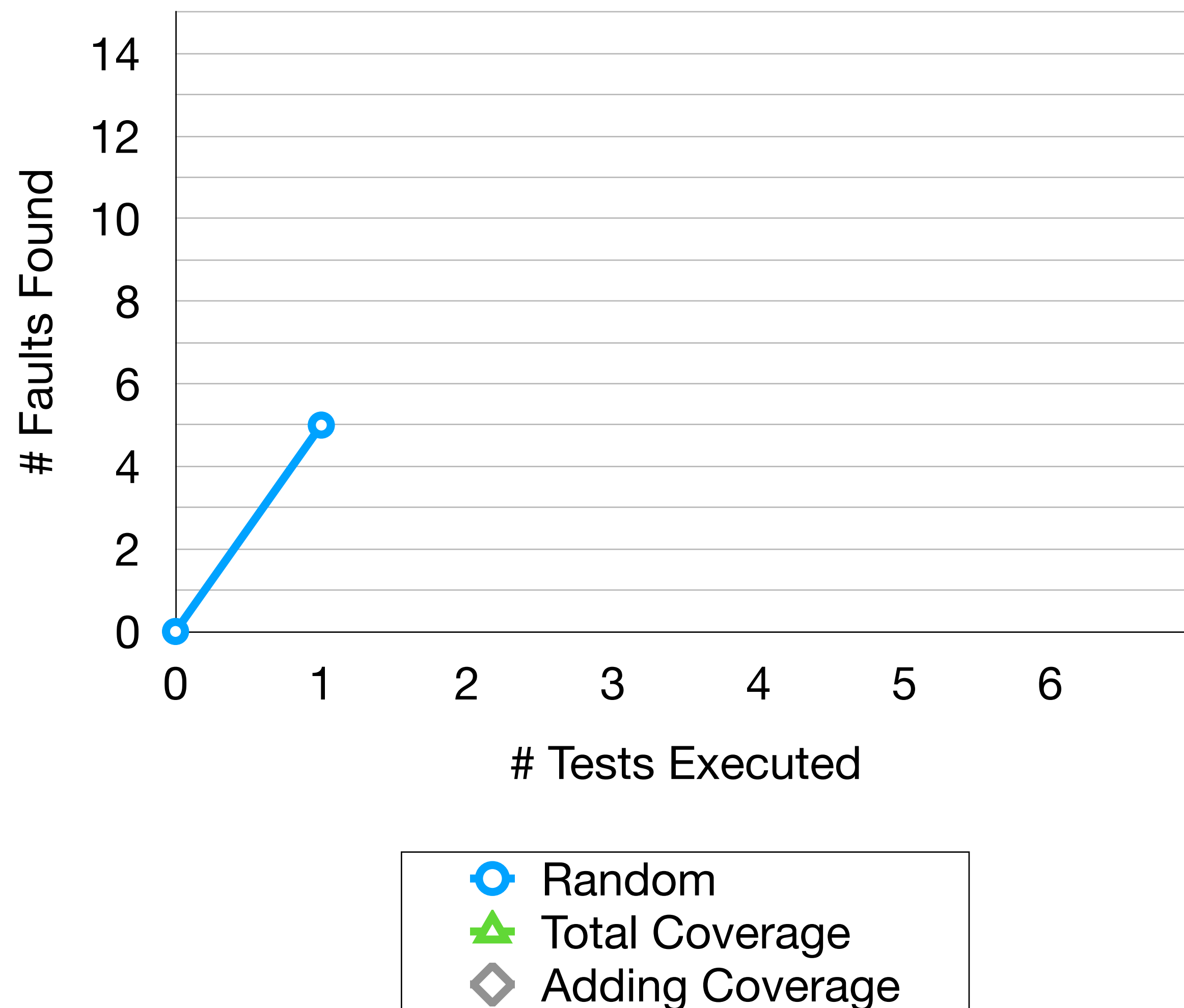
Use Mutation Testing data!

15 mutants in total

Tests A-F kill 13 of them

Random: E-A-B-F-D-C

Using Fault-Detection Data



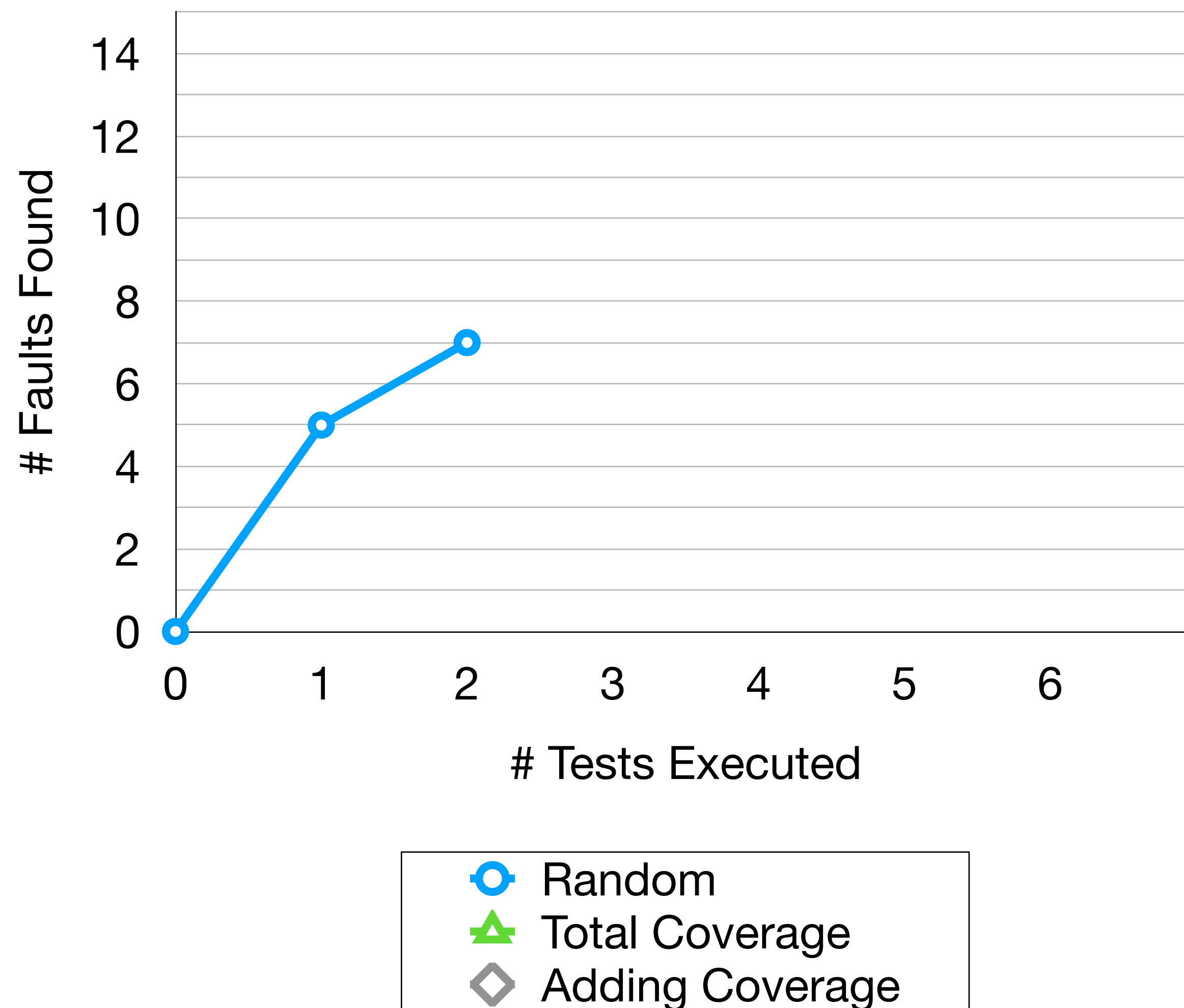
Use Mutation Testing data!

15 mutants in total

Tests A-F kill 13 of them

Random: E-A-B-F-D-C

Using Fault-Detection Data



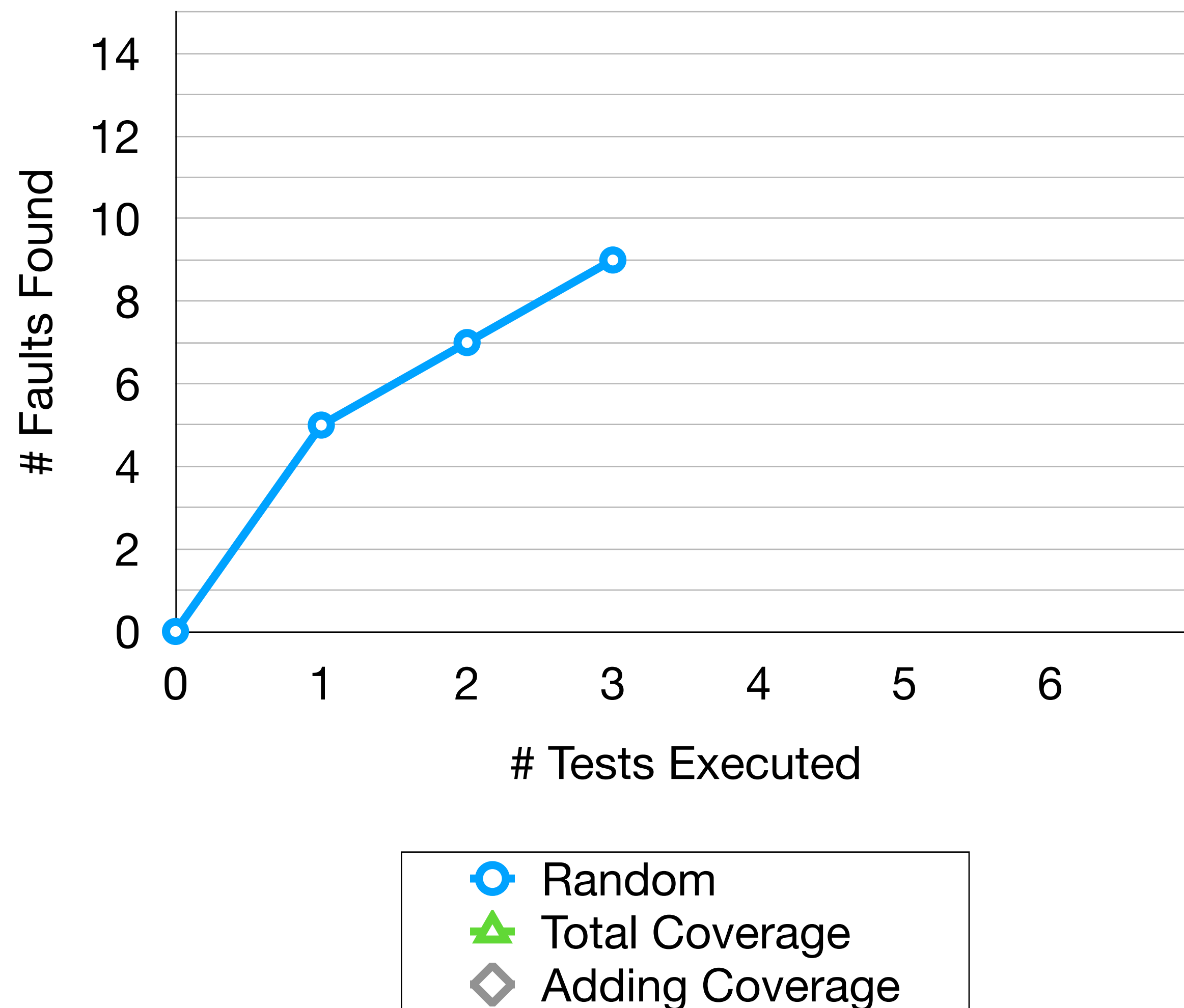
Use Mutation Testing data!

15 mutants in total

Tests A-F kill 13 of them

Random: E-A-B-F-D-C

Using Fault-Detection Data



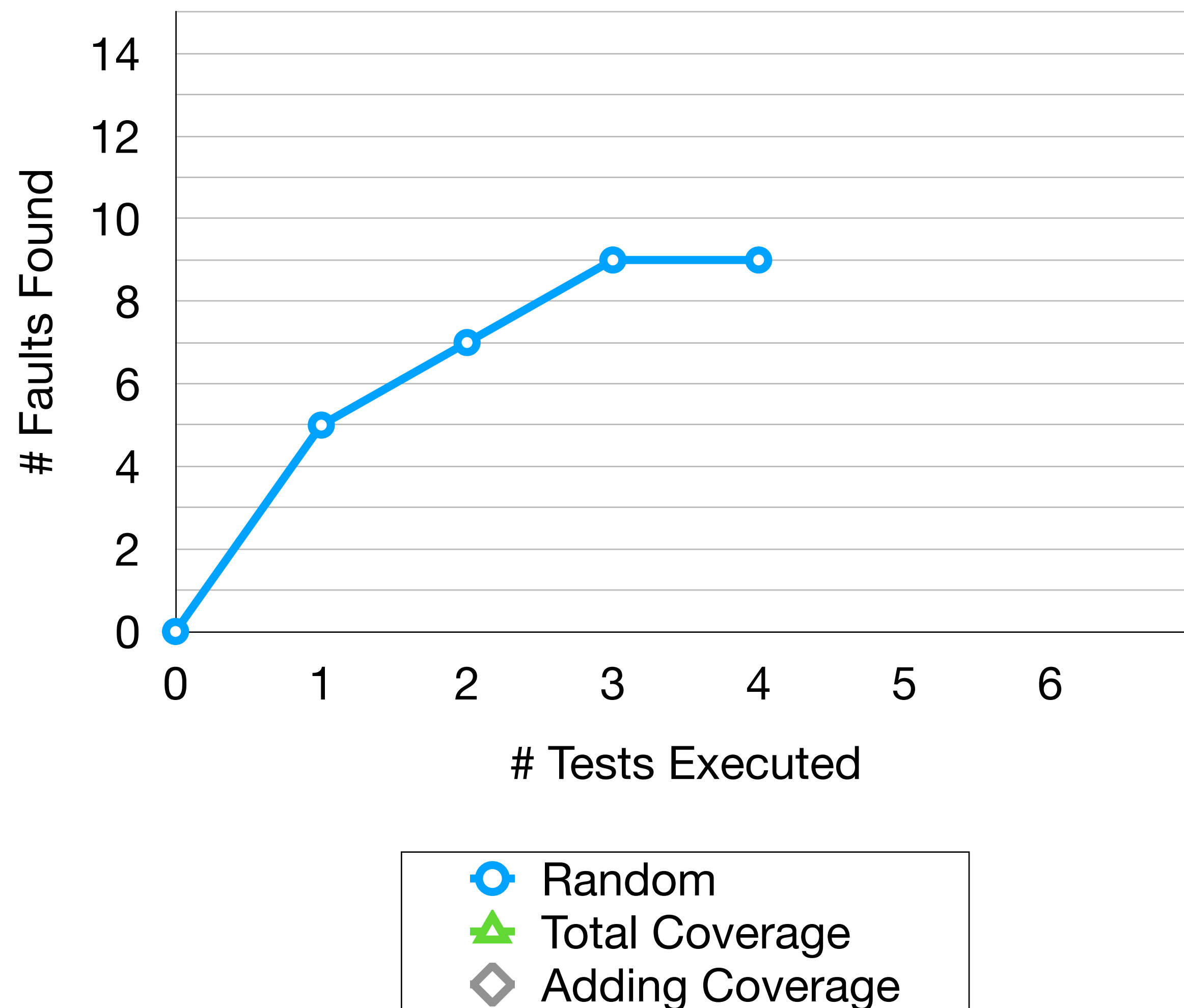
Use Mutation Testing data!

15 mutants in total

Tests A-F kill 13 of them

Random: E-A-B-F-D-C

Using Fault-Detection Data



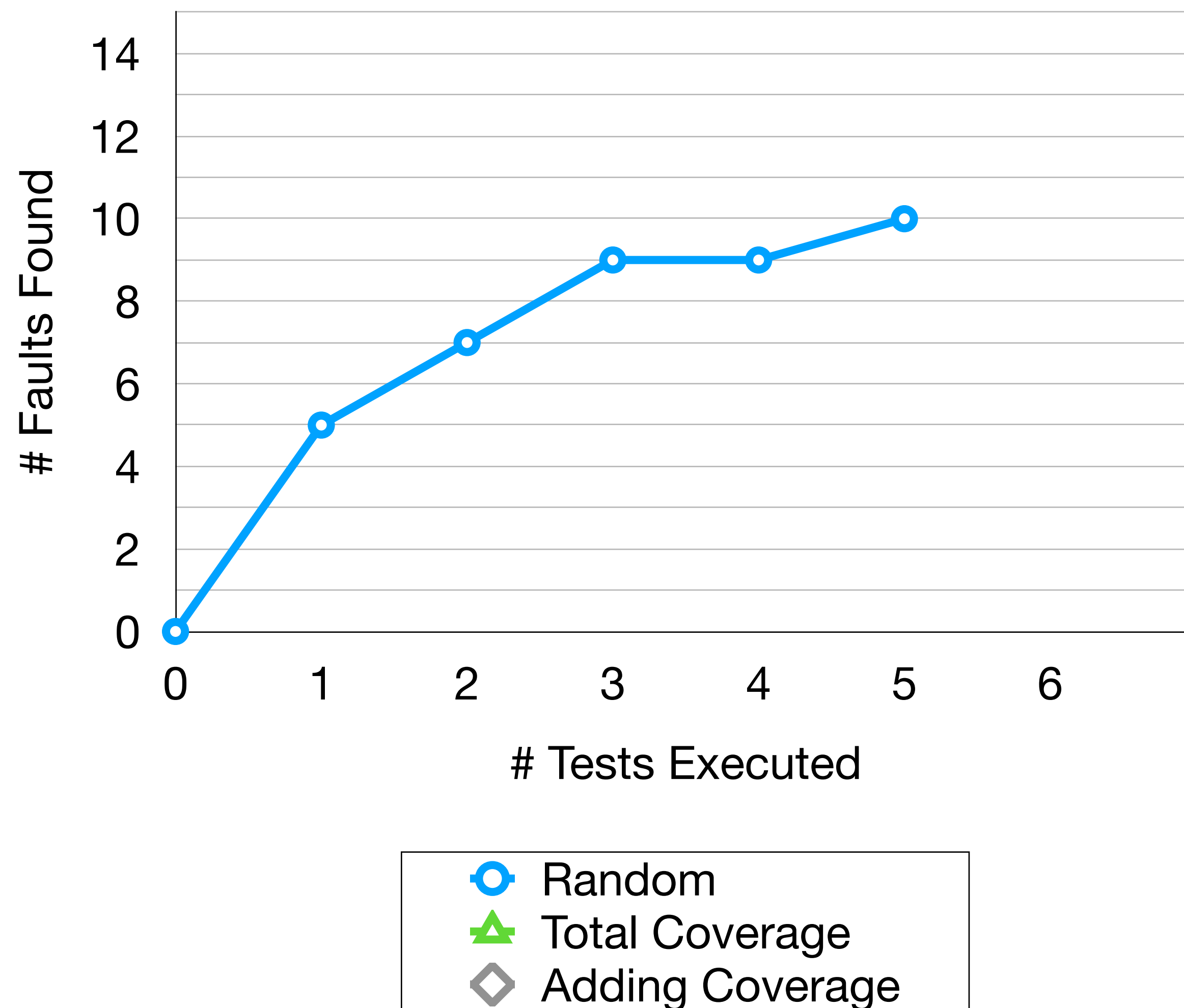
Use Mutation Testing data!

15 mutants in total

Tests A-F kill 13 of them

Random: E-A-B-F-D-C

Using Fault-Detection Data



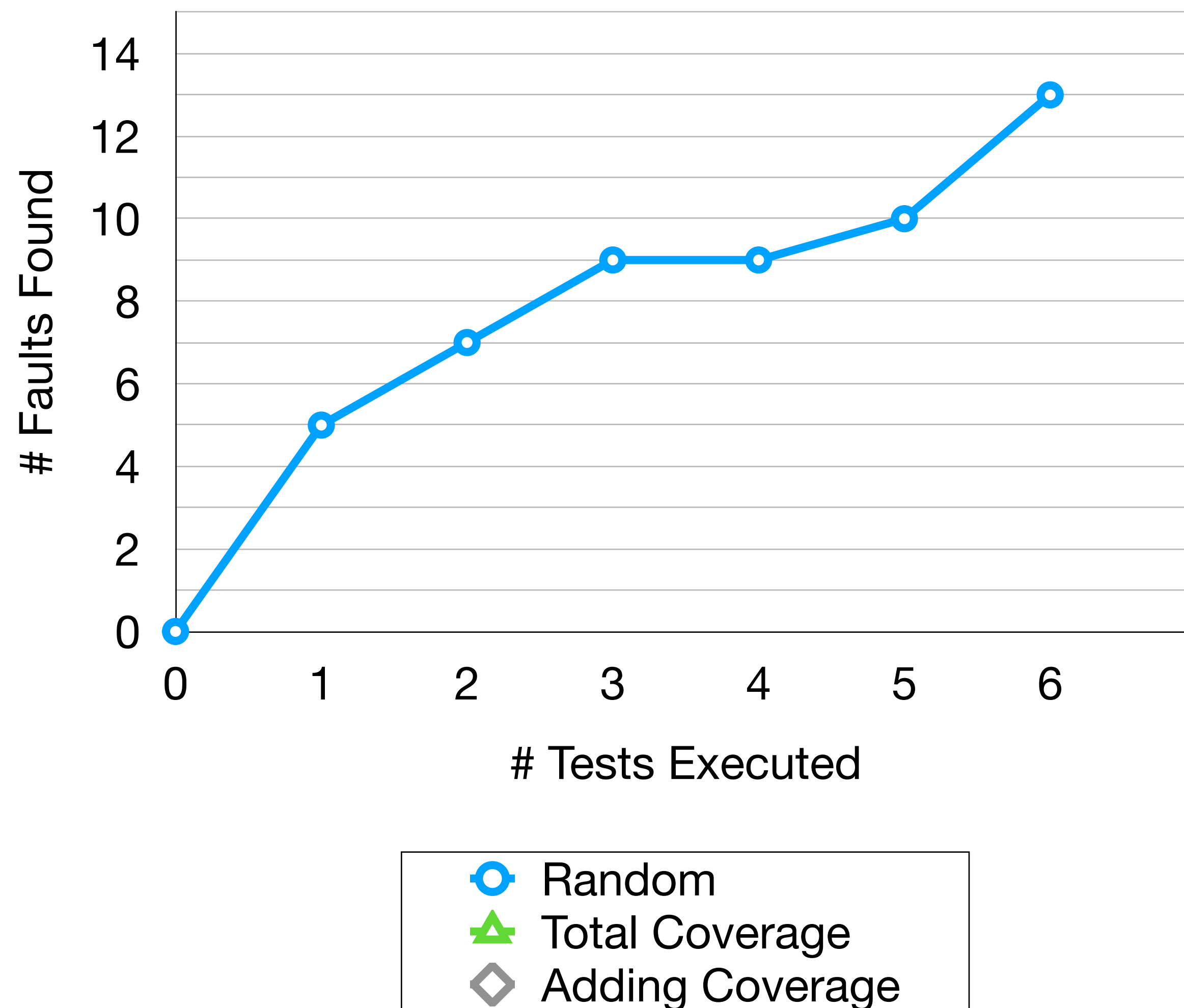
Use Mutation Testing data!

15 mutants in total

Tests A-F kill 13 of them

Random: E-A-B-F-D-C

Using Fault-Detection Data



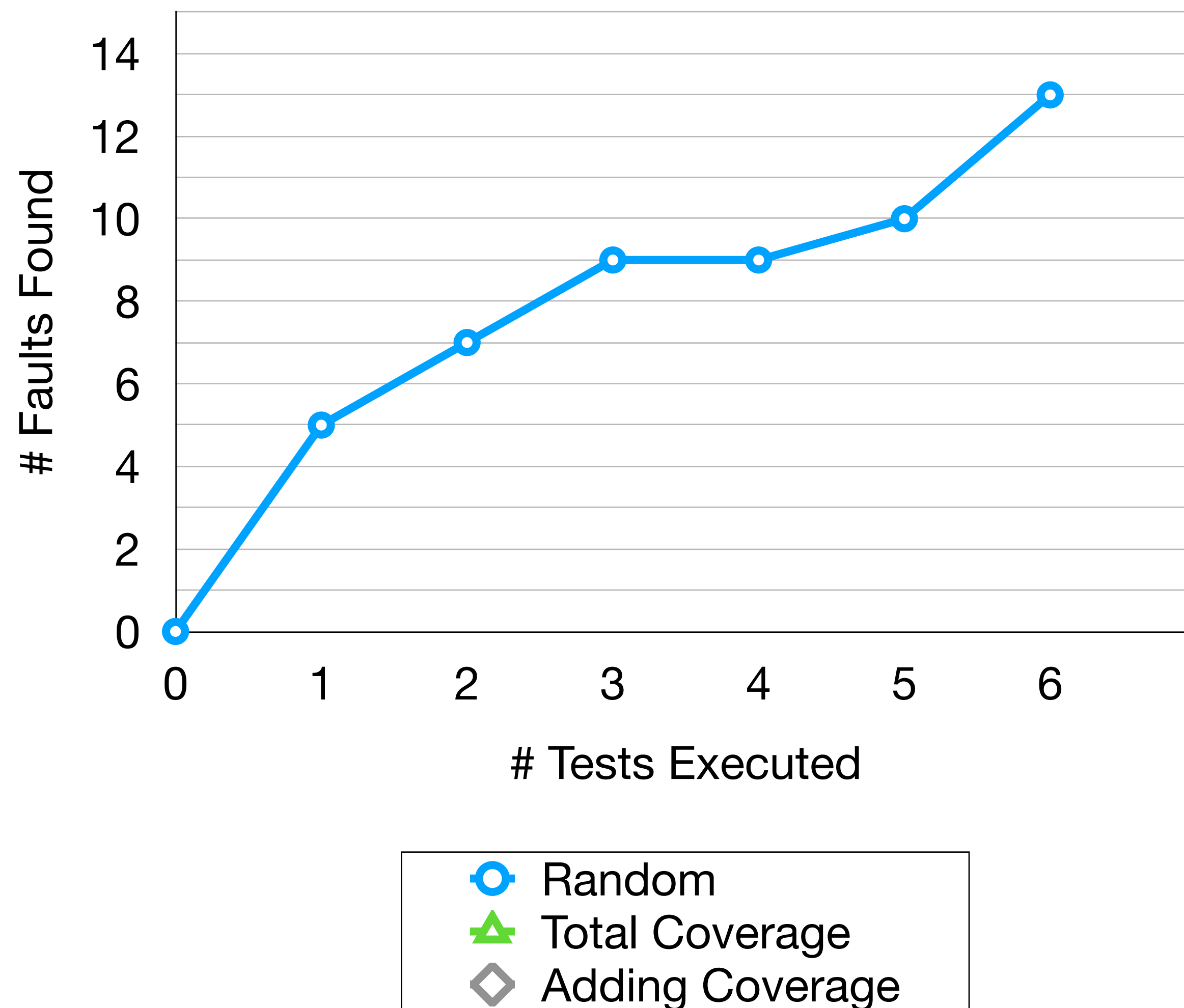
Use Mutation Testing data!

15 mutants in total

Tests A-F kill 13 of them

Random: E-A-B-F-D-C

Using Fault-Detection Data



Use Mutation Testing data!

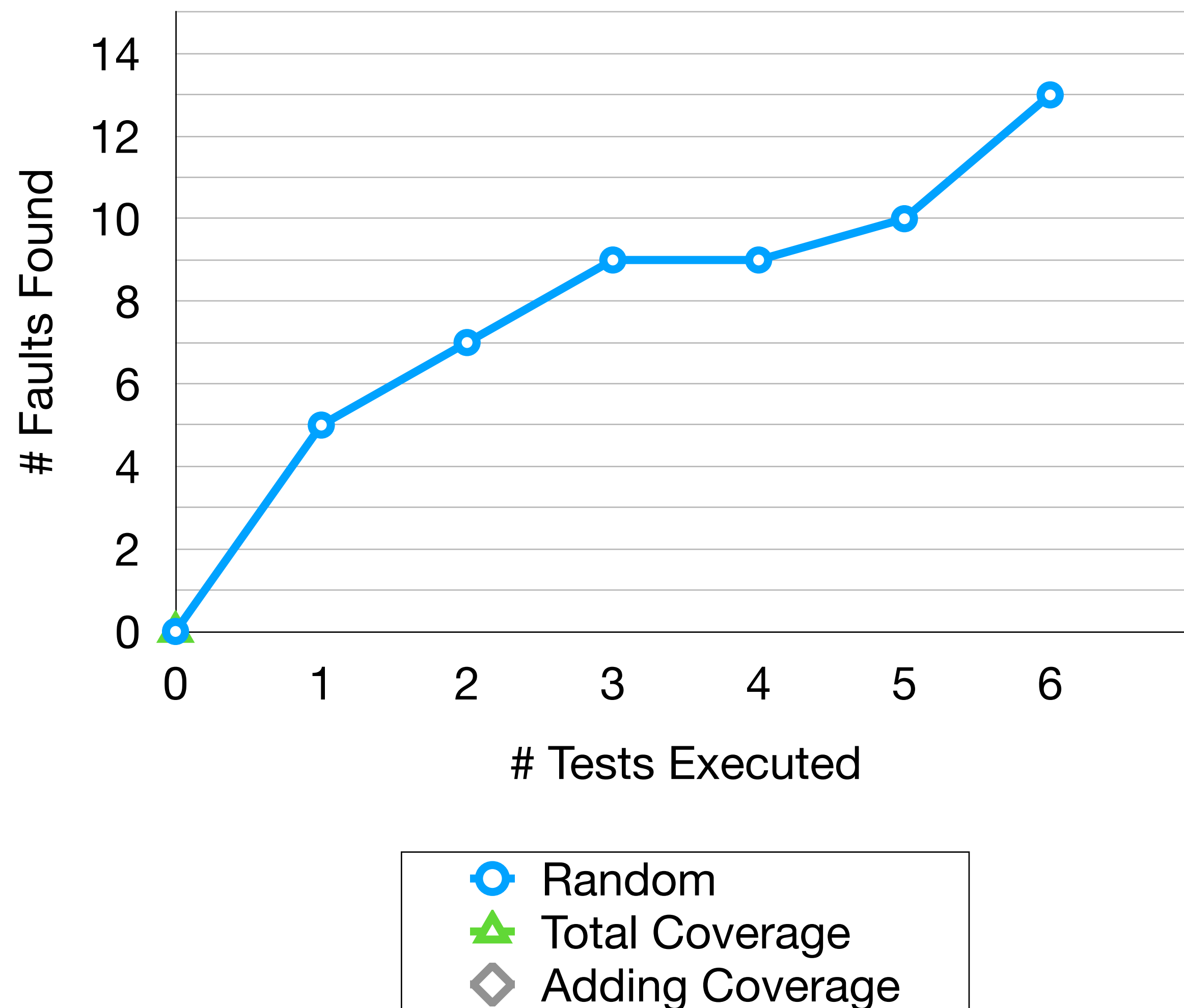
15 mutants in total

Tests A-F kill 13 of them

Random: E-A-B-F-D-C

Total Cov: C-E-D-B-F-A

Using Fault-Detection Data



Use Mutation Testing data!

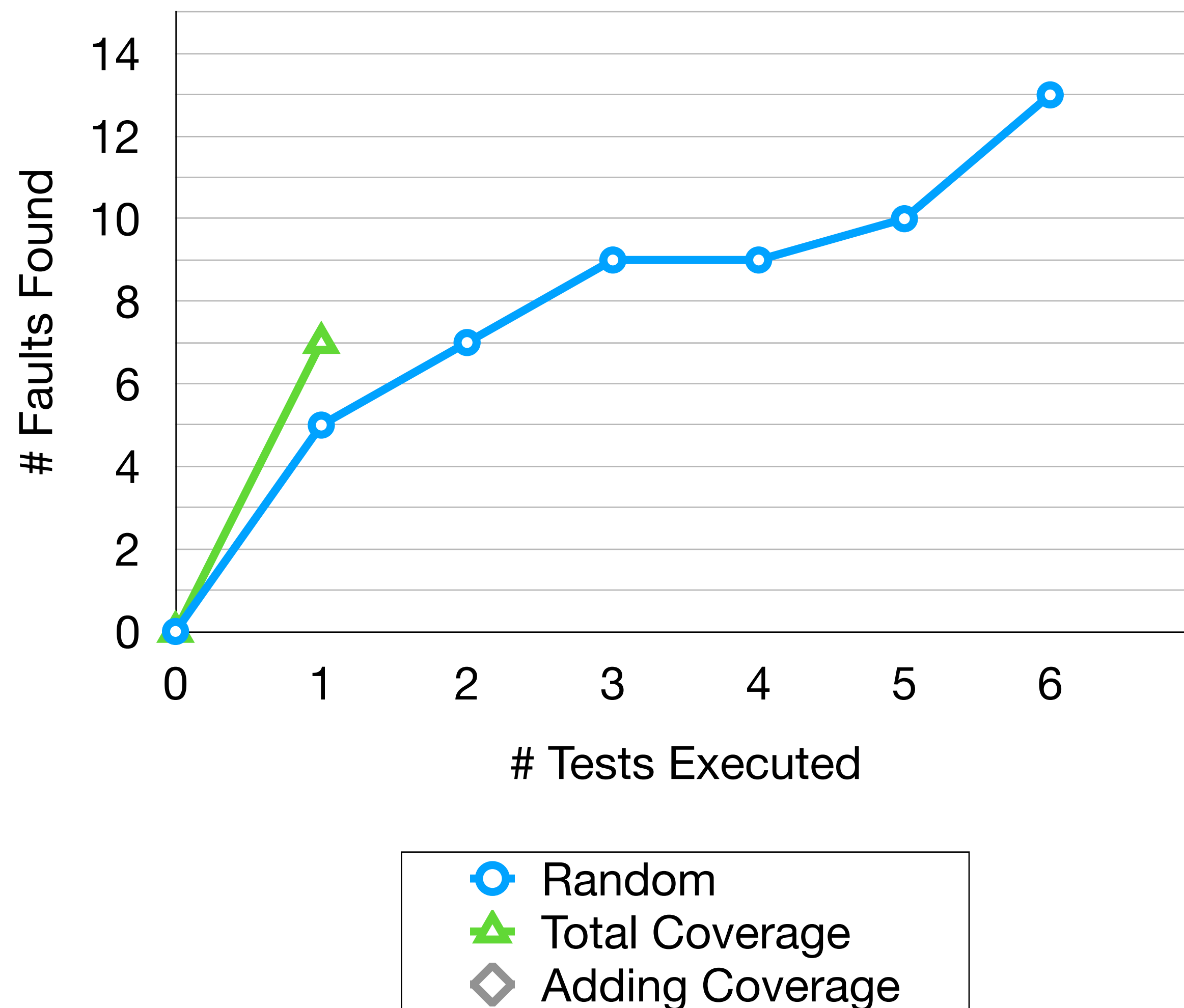
15 mutants in total

Tests A-F kill 13 of them

Random: E-A-B-F-D-C

Total Cov: C-E-D-B-F-A

Using Fault-Detection Data



Use Mutation Testing data!

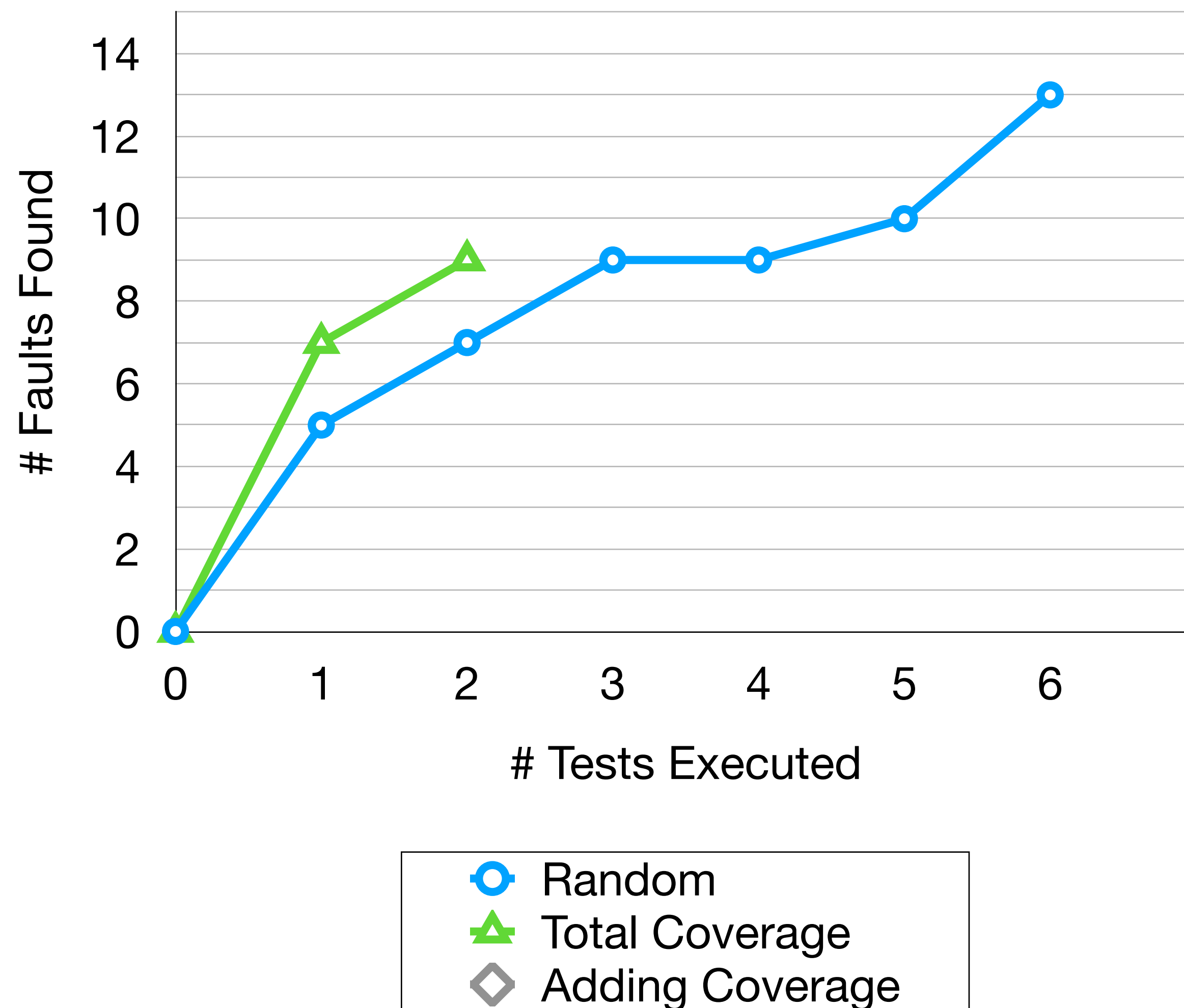
15 mutants in total

Tests A-F kill 13 of them

Random: E-A-B-F-D-C

Total Cov: C-E-D-B-F-A

Using Fault-Detection Data



Use Mutation Testing data!

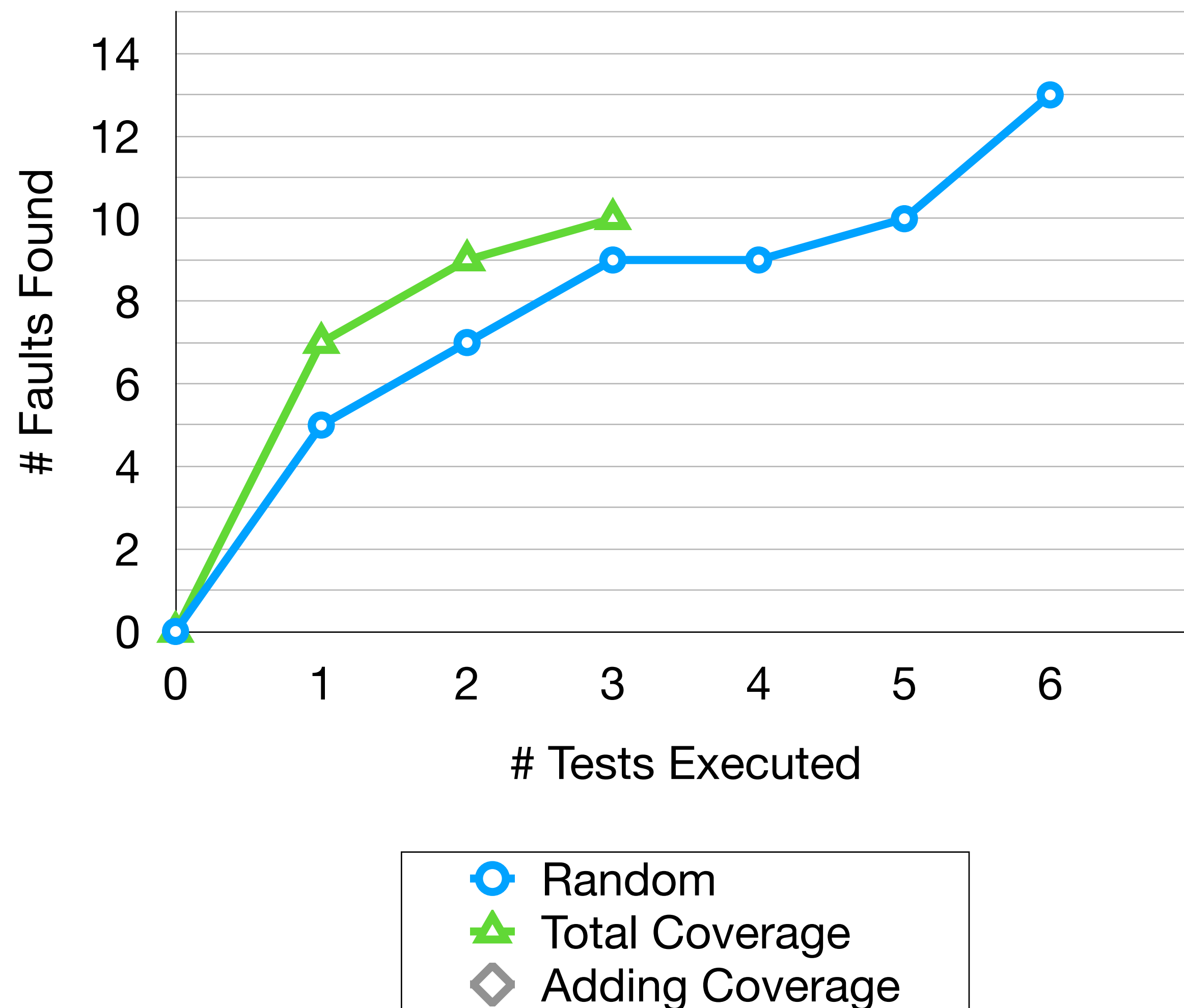
15 mutants in total

Tests A-F kill 13 of them

Random: E-A-B-F-D-C

Total Cov: C-E-D-B-F-A

Using Fault-Detection Data



Use Mutation Testing data!

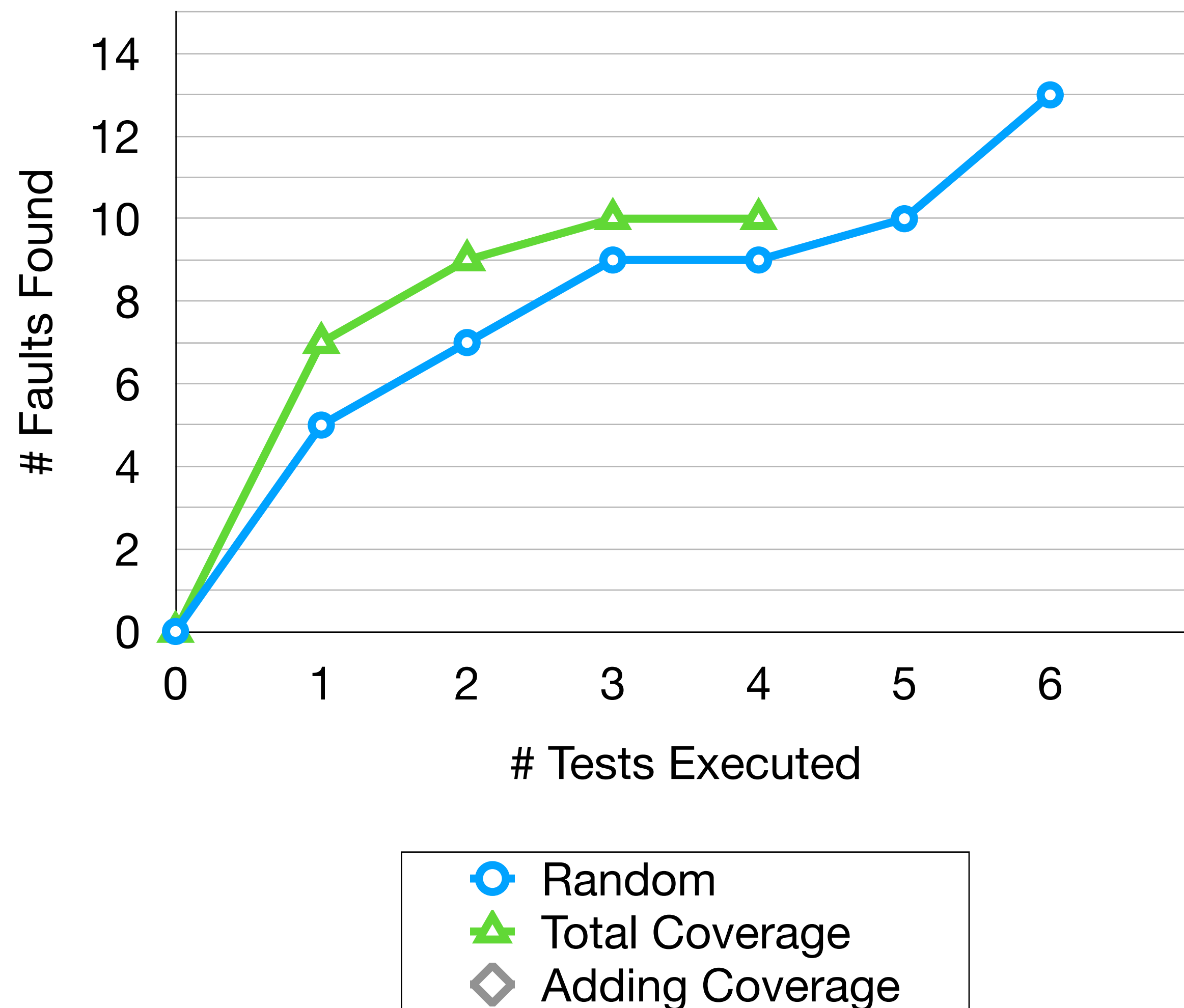
15 mutants in total

Tests A-F kill 13 of them

Random: E-A-B-F-D-C

Total Cov: C-E-D-B-F-A

Using Fault-Detection Data



Use Mutation Testing data!

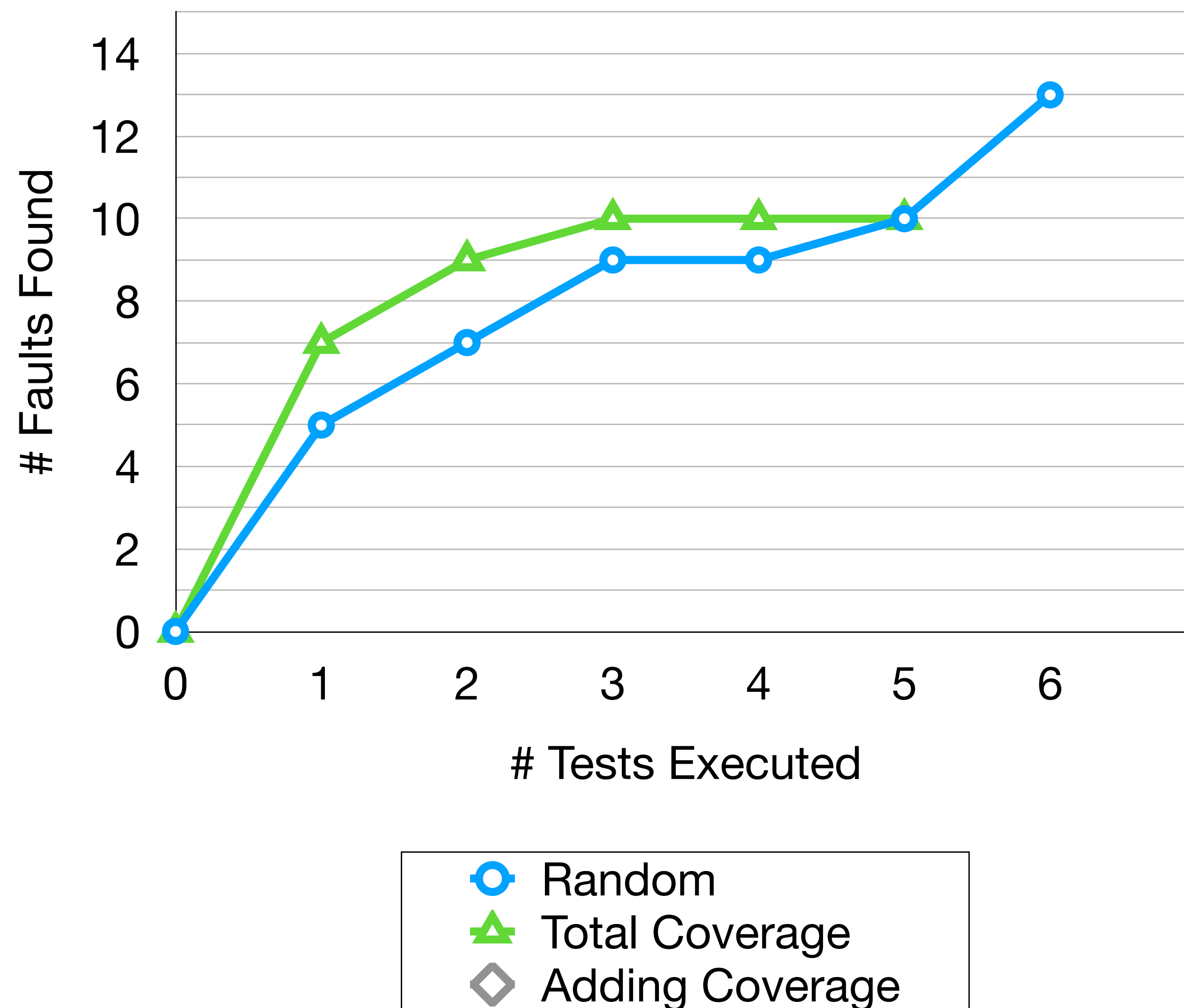
15 mutants in total

Tests A-F kill 13 of them

Random: E-A-B-F-D-C

Total Cov: C-E-D-B-F-A

Using Fault-Detection Data



Use Mutation Testing data!

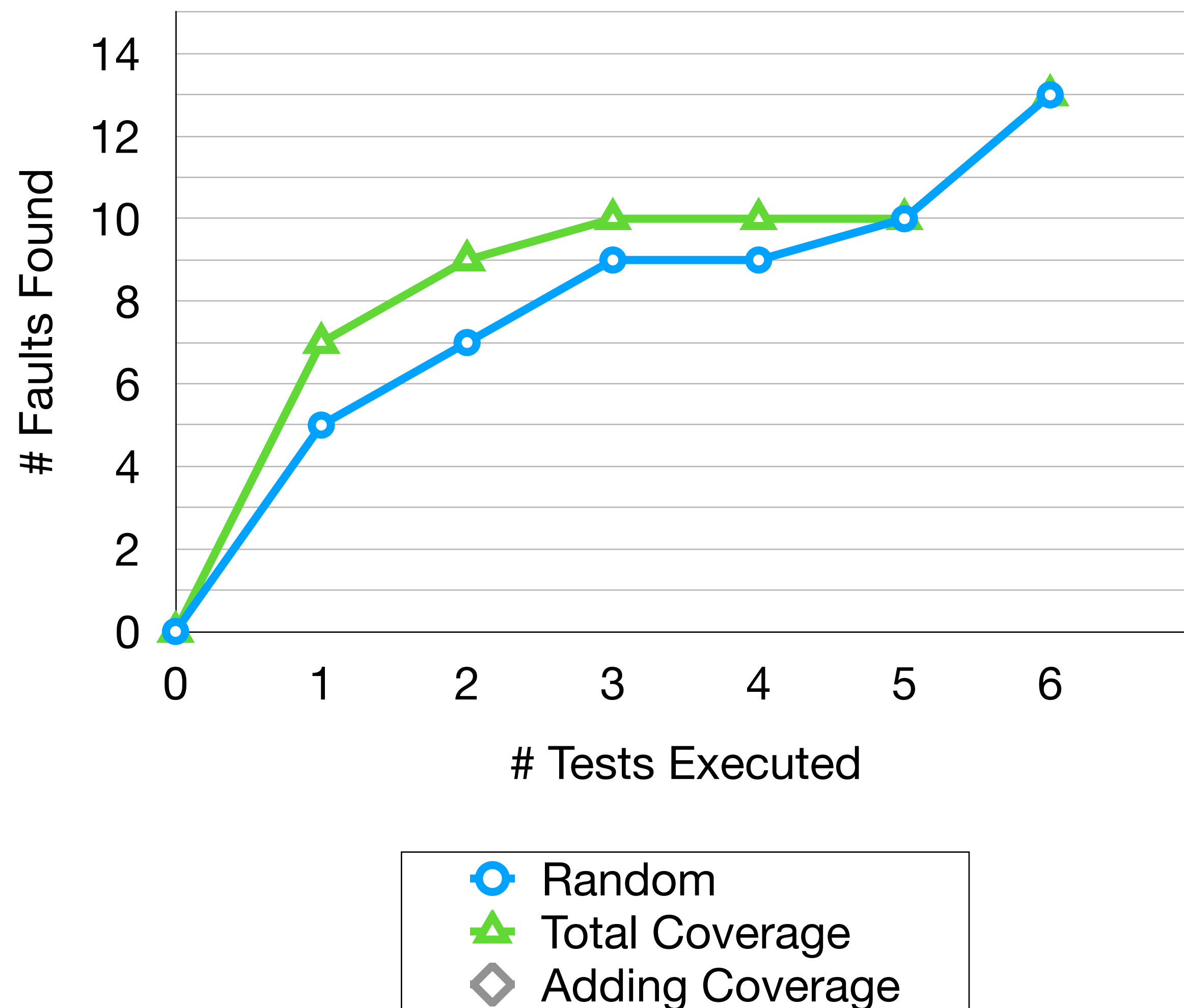
15 mutants in total

Tests A-F kill 13 of them

Random: E-A-B-F-D-C

Total Cov: C-E-D-B-F-A

Using Fault-Detection Data



Use Mutation Testing data!

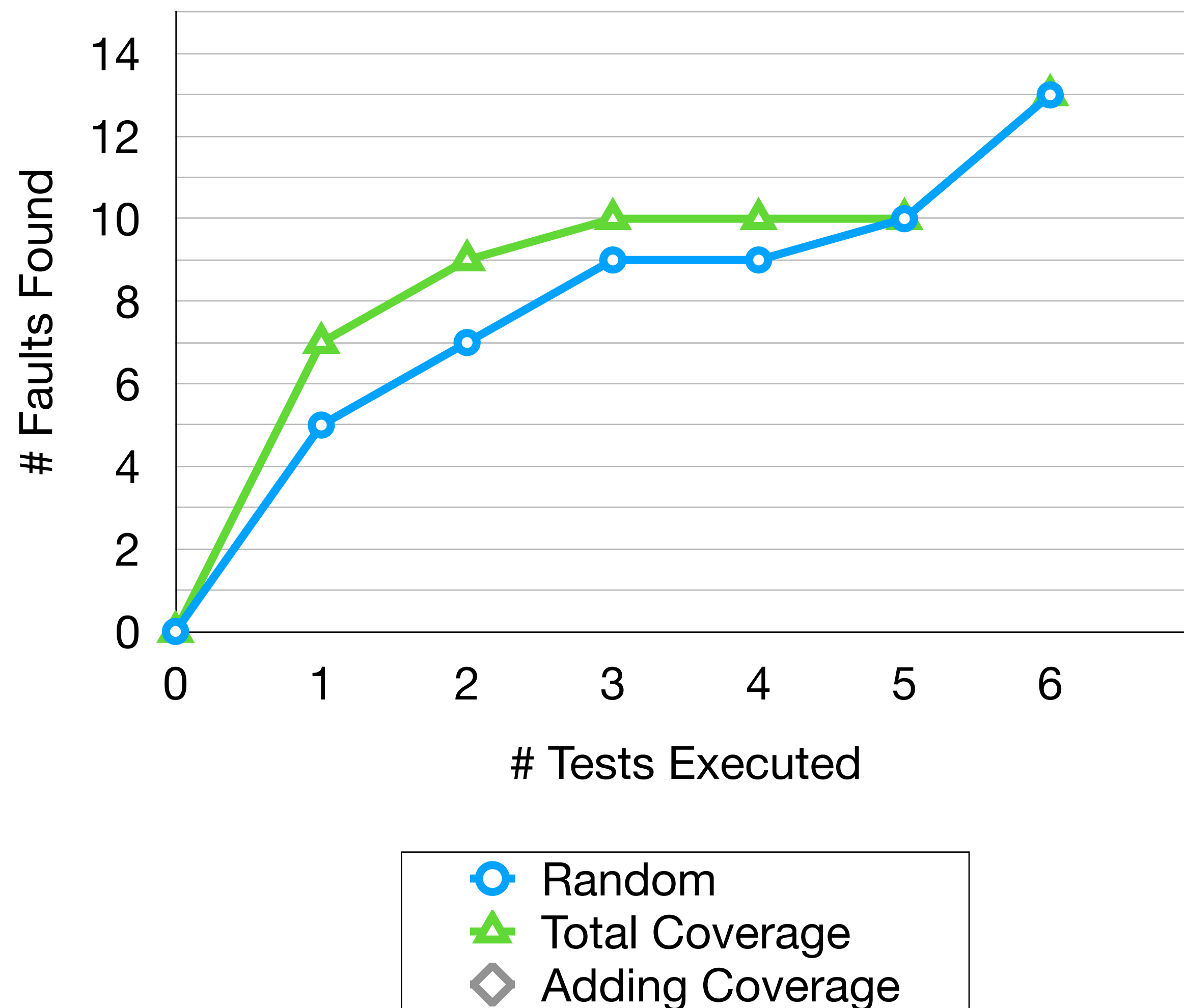
15 mutants in total

Tests A-F kill 13 of them

Random: E-A-B-F-D-C

Total Cov: C-E-D-B-F-A

Using Fault-Detection Data



Use Mutation Testing data!

15 mutants in total

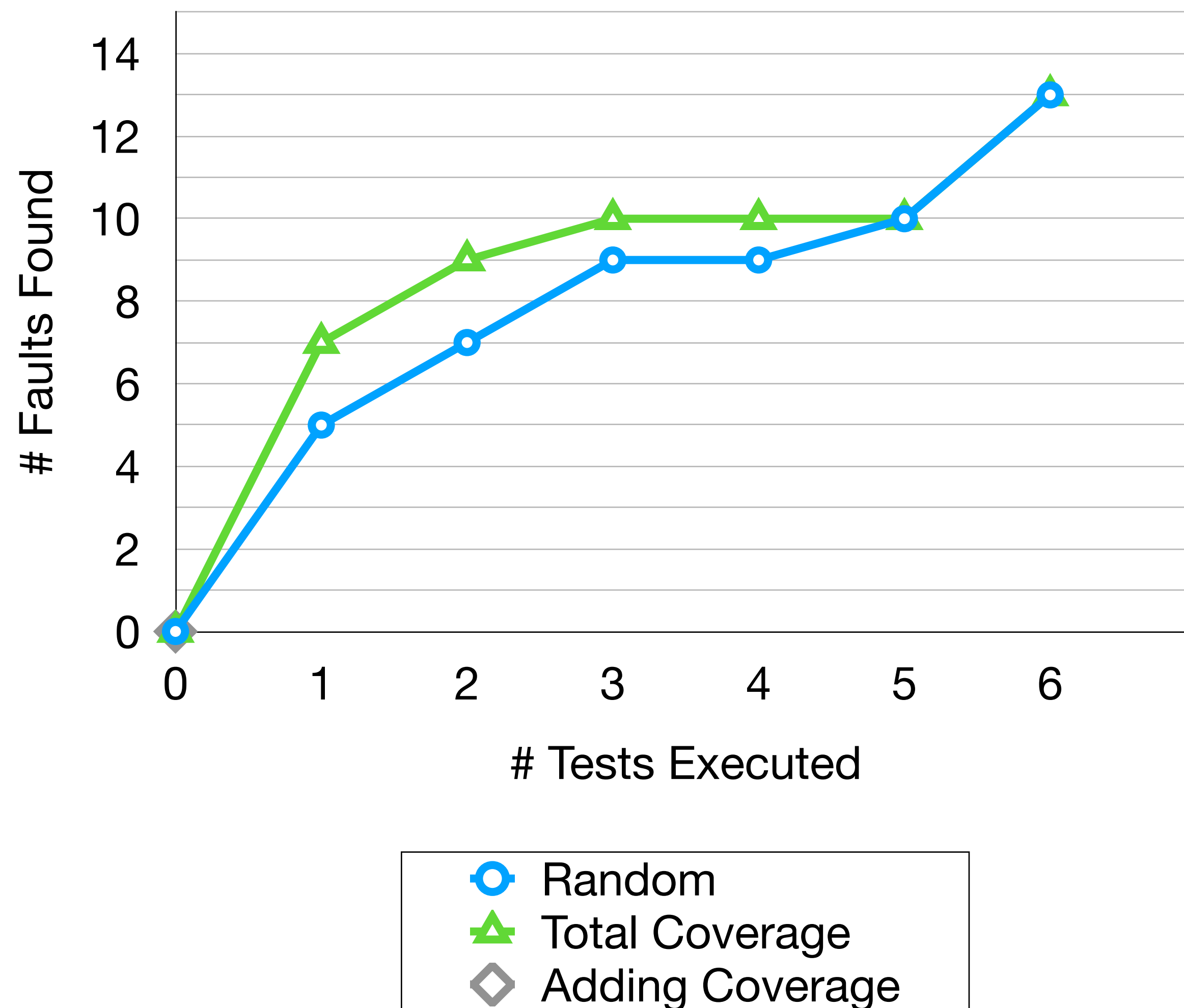
Tests A-F kill 13 of them

Random: E-A-B-F-D-C

Total Cov: C-E-D-B-F-A

Adding Cov: C-E-A-F-D-B

Using Fault-Detection Data



Use Mutation Testing data!

15 mutants in total

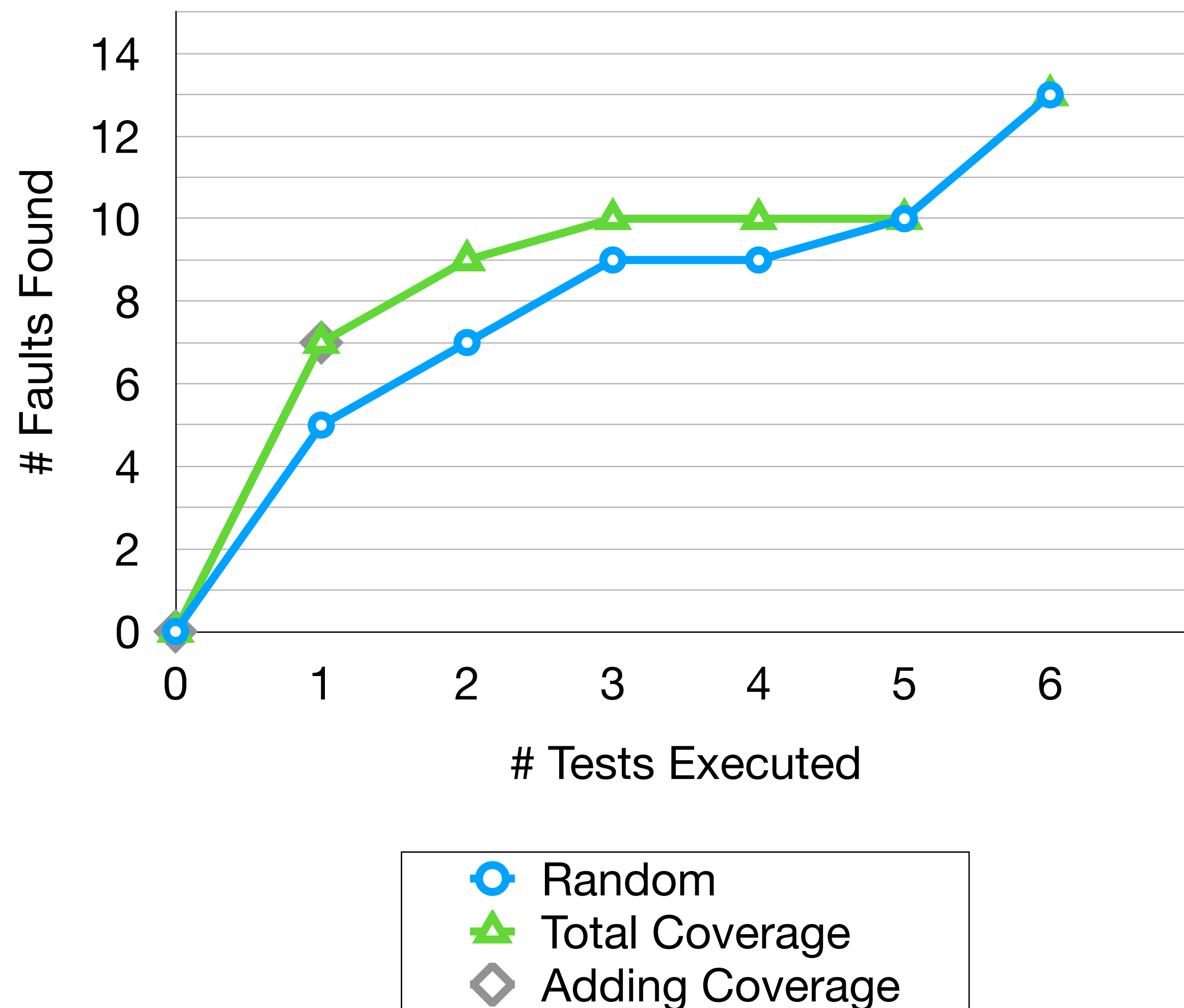
Tests A-F kill 13 of them

Random: E-A-B-F-D-C

Total Cov: C-E-D-B-F-A

Adding Cov: C-E-A-F-D-B

Using Fault-Detection Data



Use Mutation Testing data!

15 mutants in total

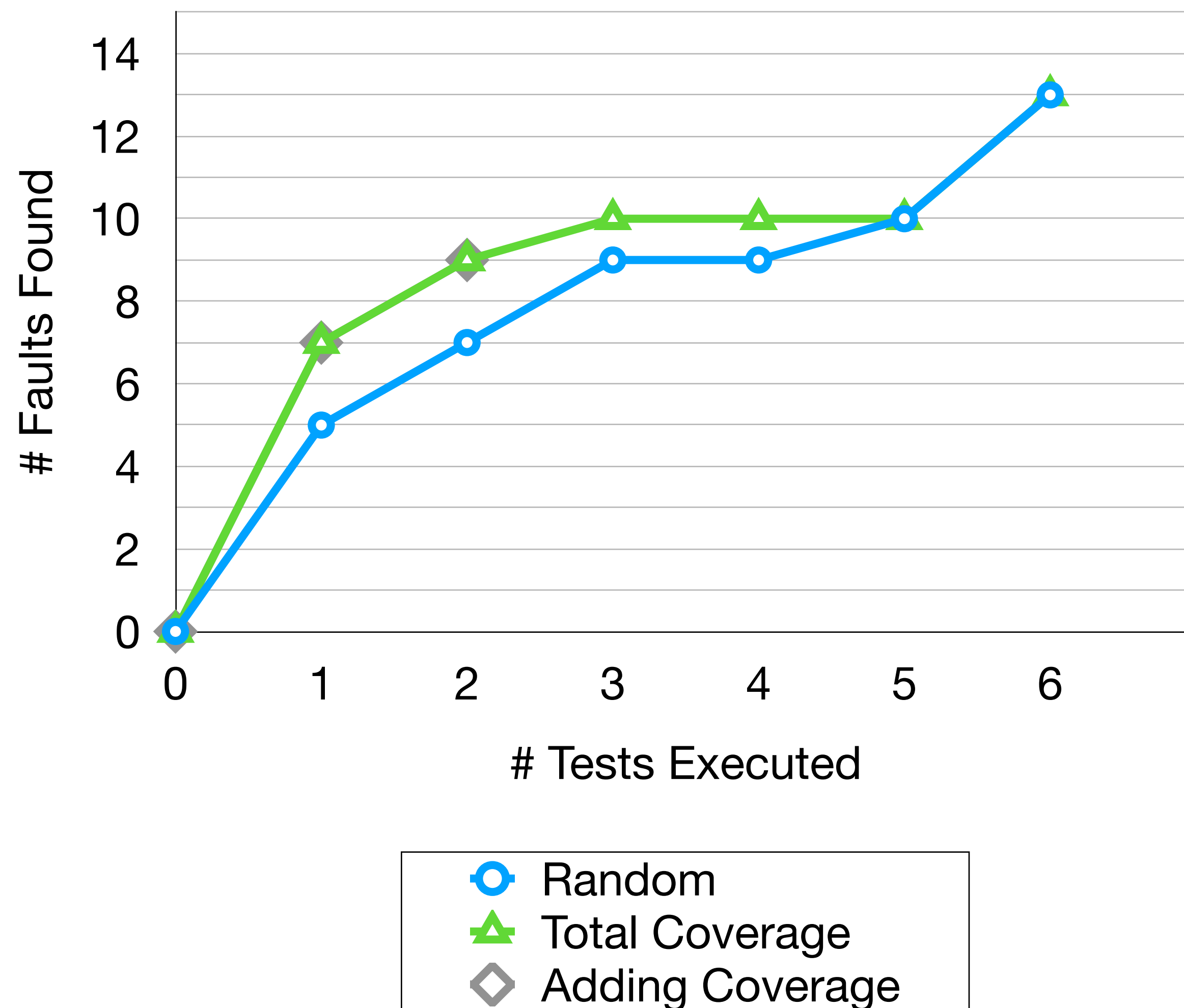
Tests A-F kill 13 of them

Random: E-A-B-F-D-C

Total Cov: C-E-D-B-F-A

Adding Cov: C-E-A-F-D-B

Using Fault-Detection Data



Use Mutation Testing data!

15 mutants in total

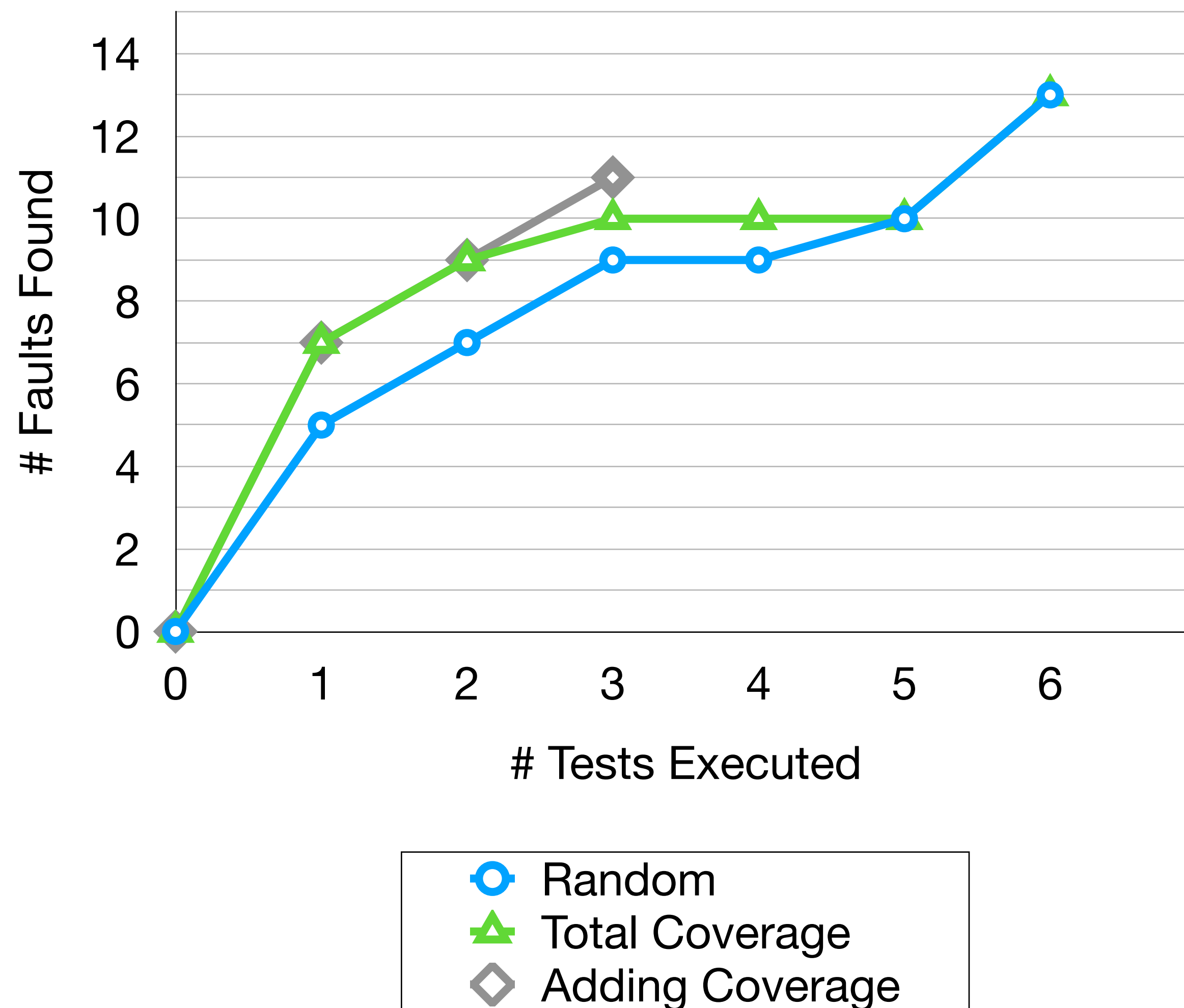
Tests A-F kill 13 of them

Random: E-A-B-F-D-C

Total Cov: C-E-D-B-F-A

Adding Cov: C-E-A-F-D-B

Using Fault-Detection Data



Use Mutation Testing data!

15 mutants in total

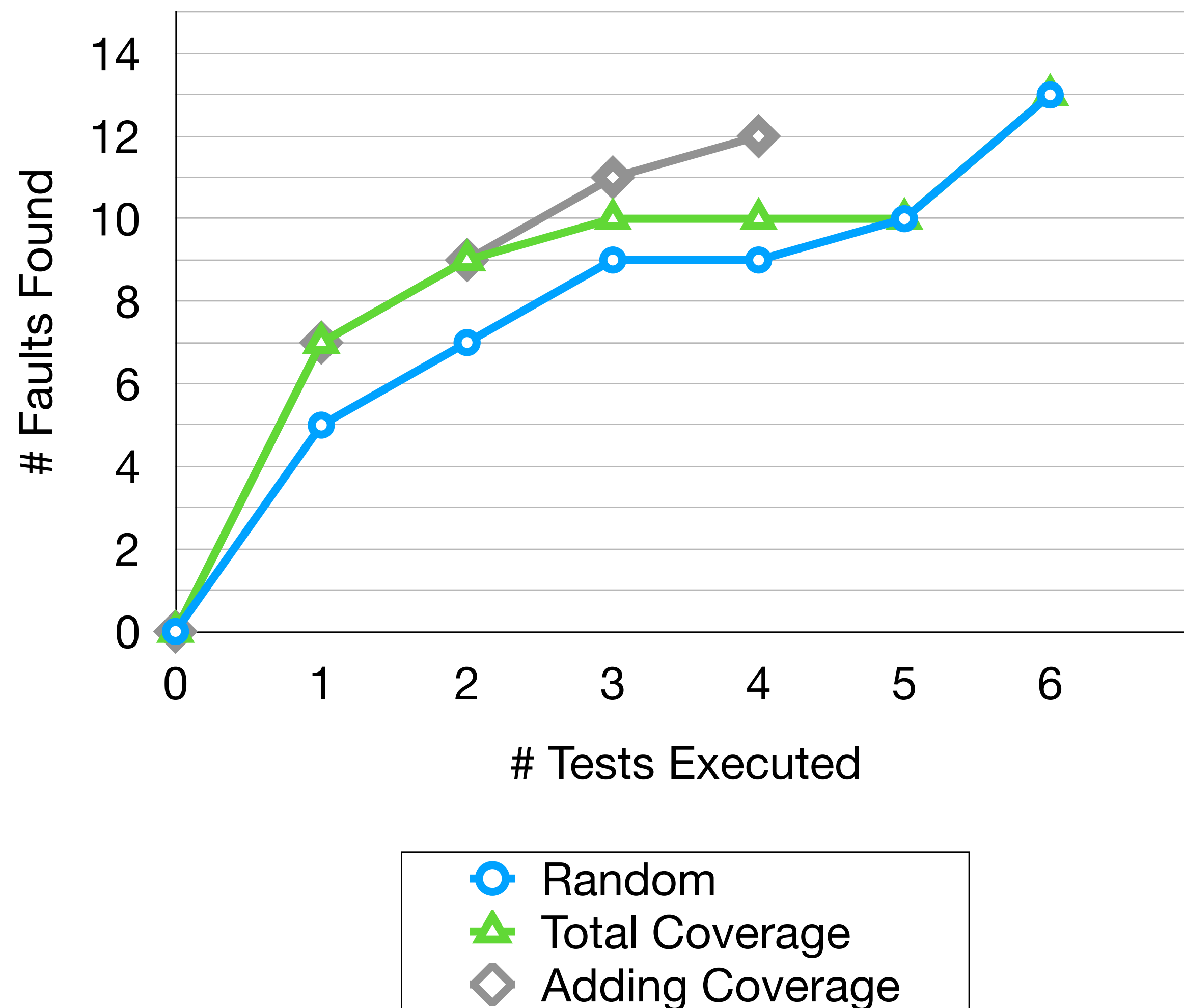
Tests A-F kill 13 of them

Random: E-A-B-F-D-C

Total Cov: C-E-D-B-F-A

Adding Cov: C-E-A-F-D-B

Using Fault-Detection Data



Use Mutation Testing data!

15 mutants in total

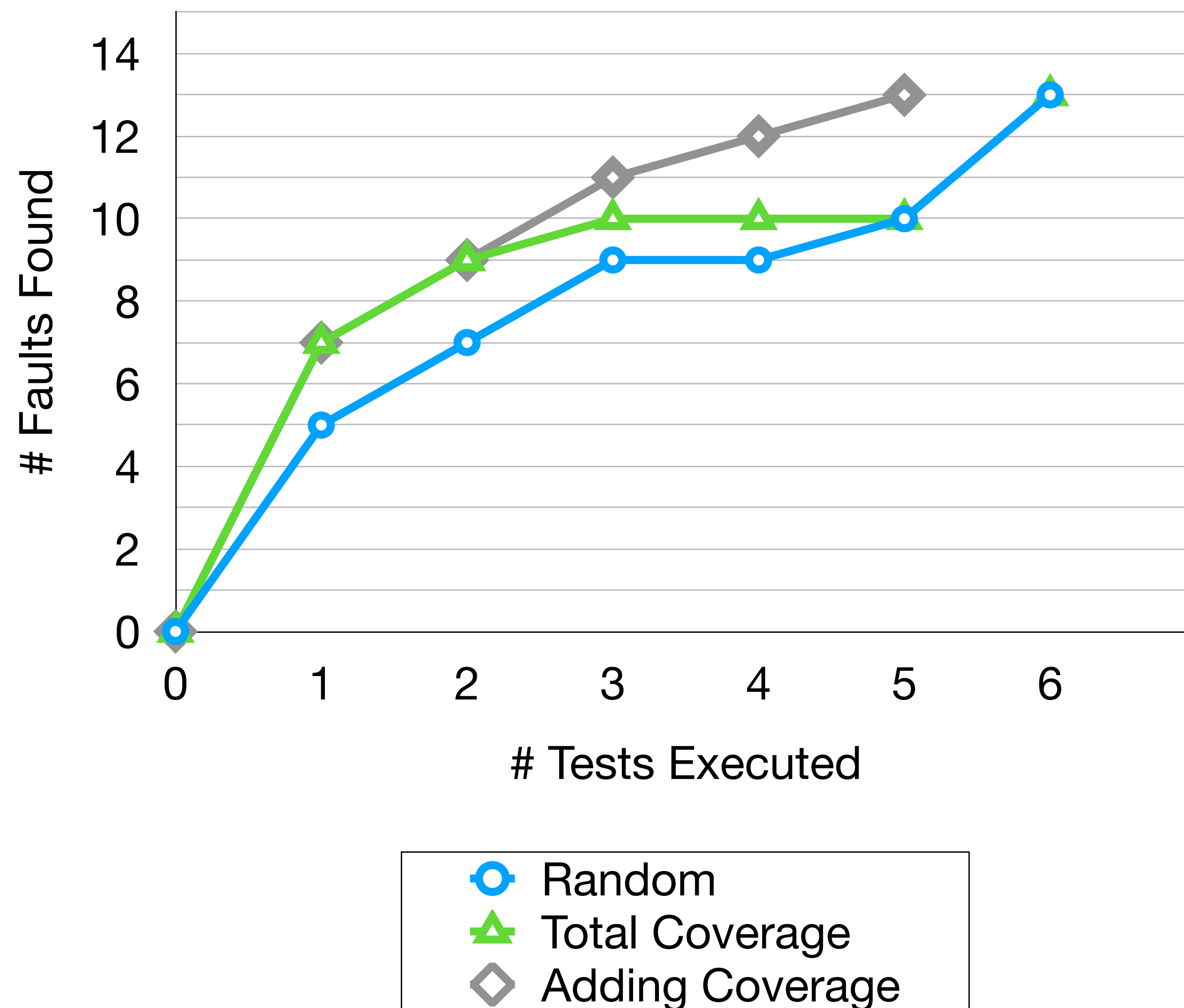
Tests A-F kill 13 of them

Random: E-A-B-F-D-C

Total Cov: C-E-D-B-F-A

Adding Cov: C-E-A-F-D-B

Using Fault-Detection Data



Use Mutation Testing data!

15 mutants in total

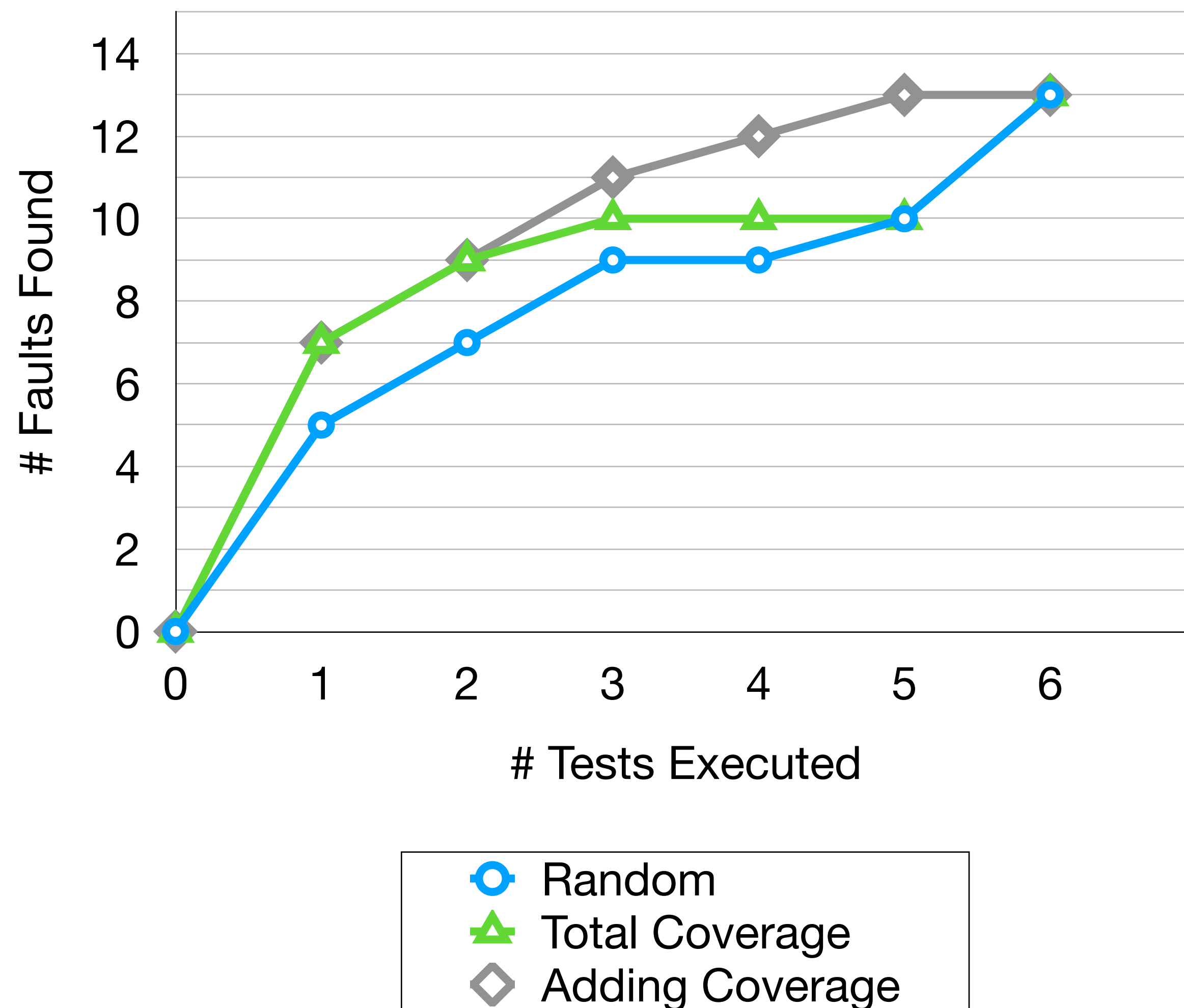
Tests A-F kill 13 of them

Random: E-A-B-F-D-C

Total Cov: C-E-D-B-F-A

Adding Cov: C-E-A-F-D-B

Using Fault-Detection Data



Use Mutation Testing data!

15 mutants in total

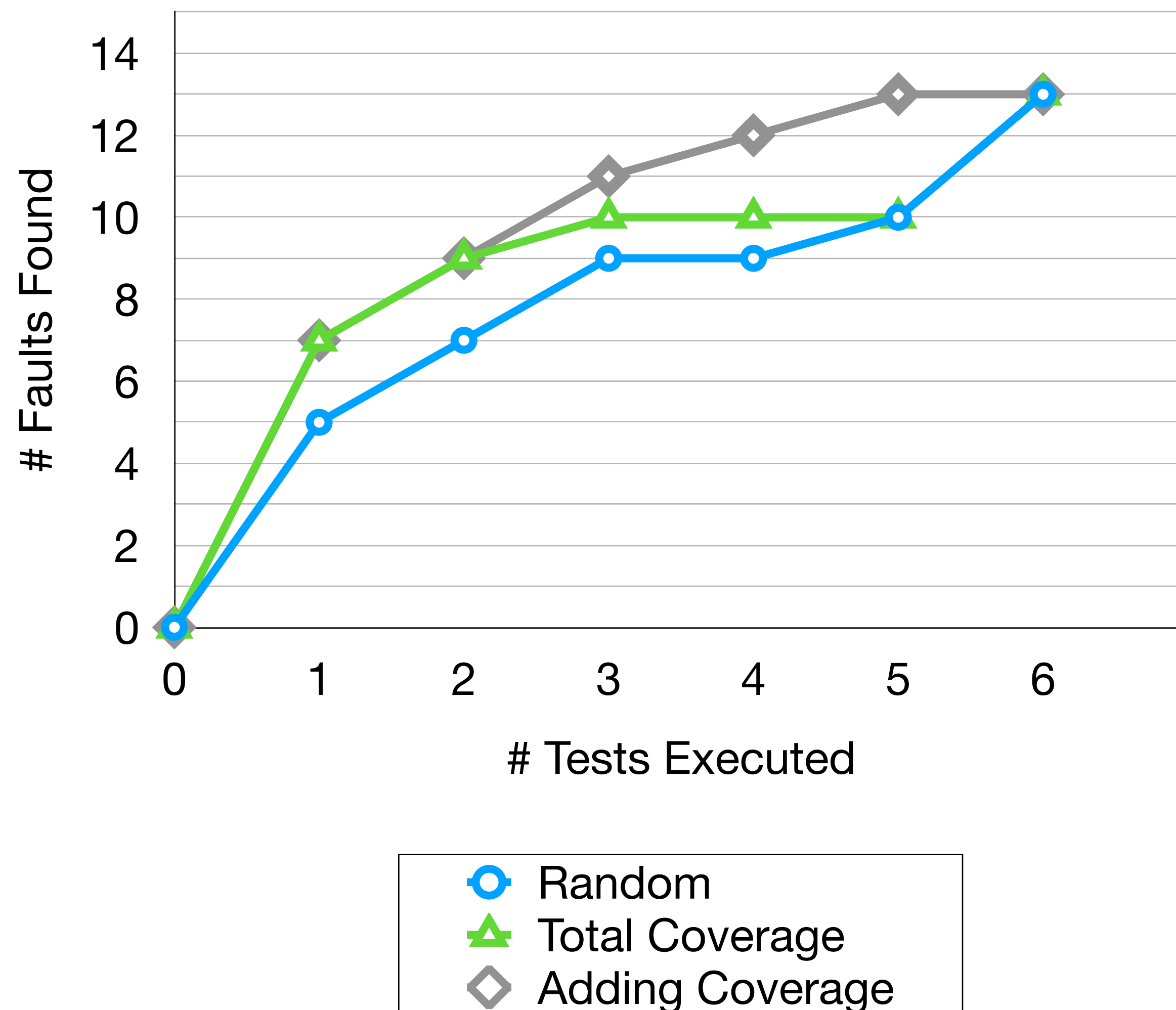
Tests A-F kill 13 of them

Random: E-A-B-F-D-C

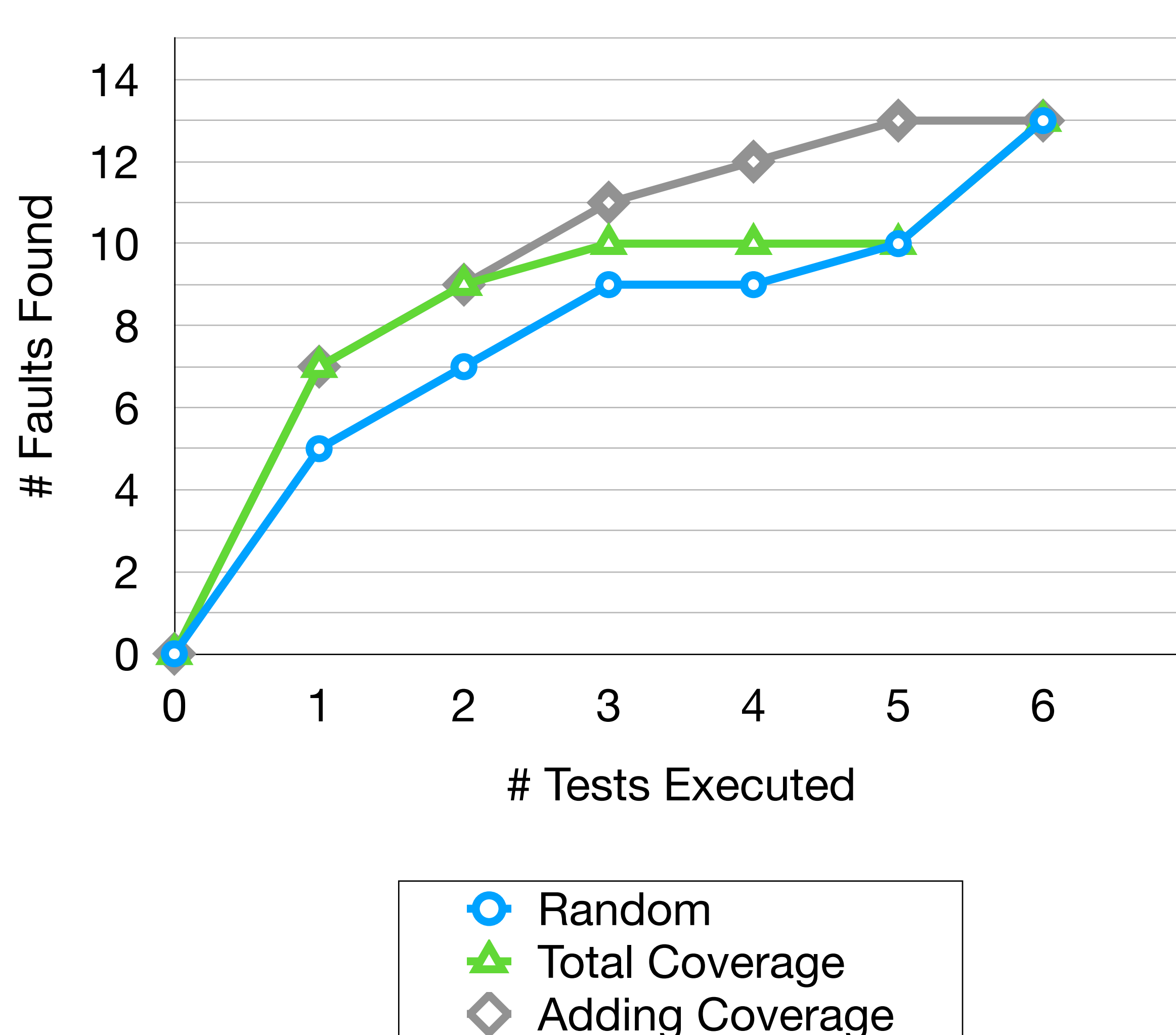
Total Cov: C-E-D-B-F-A

Adding Cov: C-E-A-F-D-B

Using Fault-Detection Data

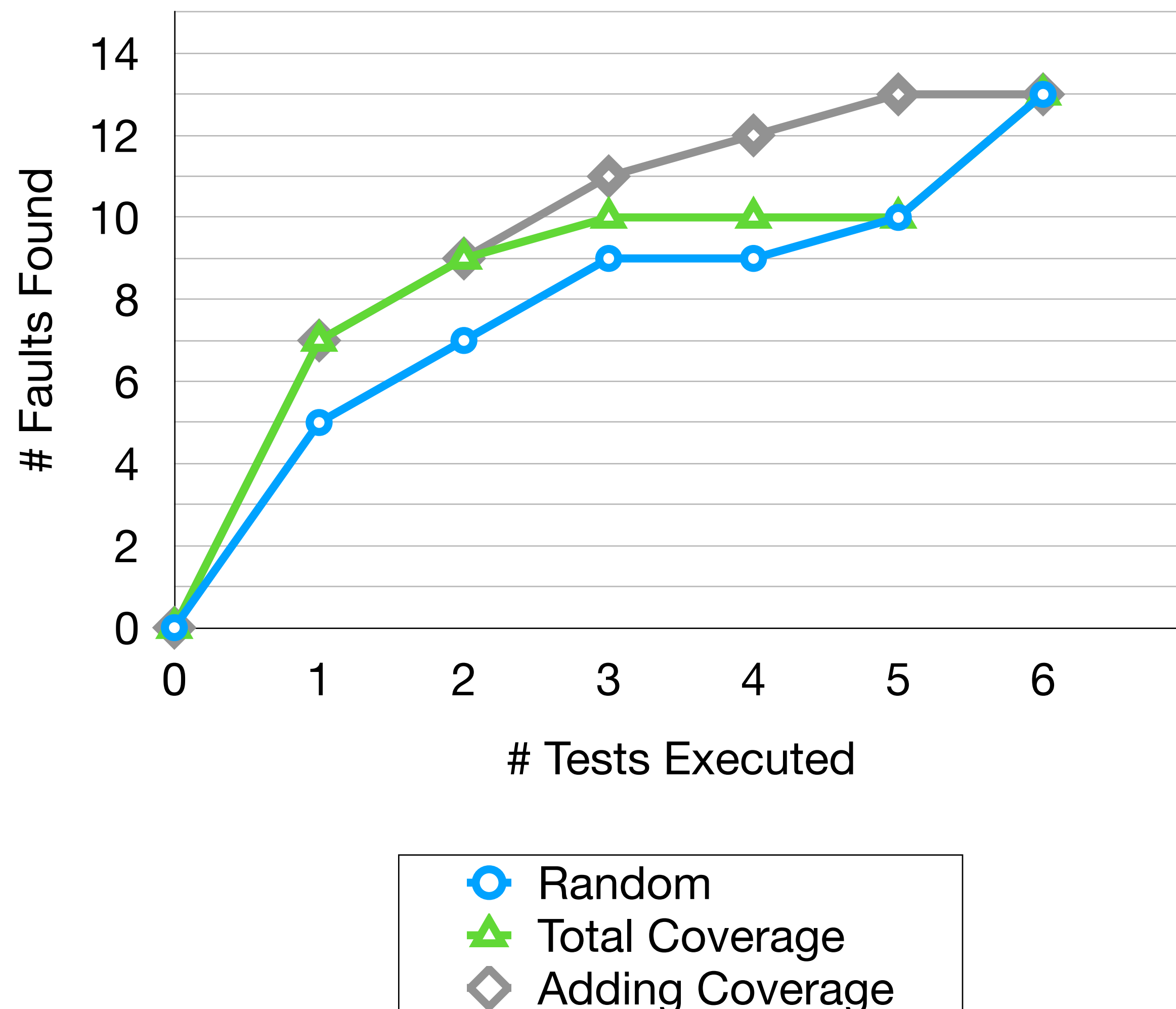


Using Fault-Detection Data



APFD: Average Percentage of Faults Detected

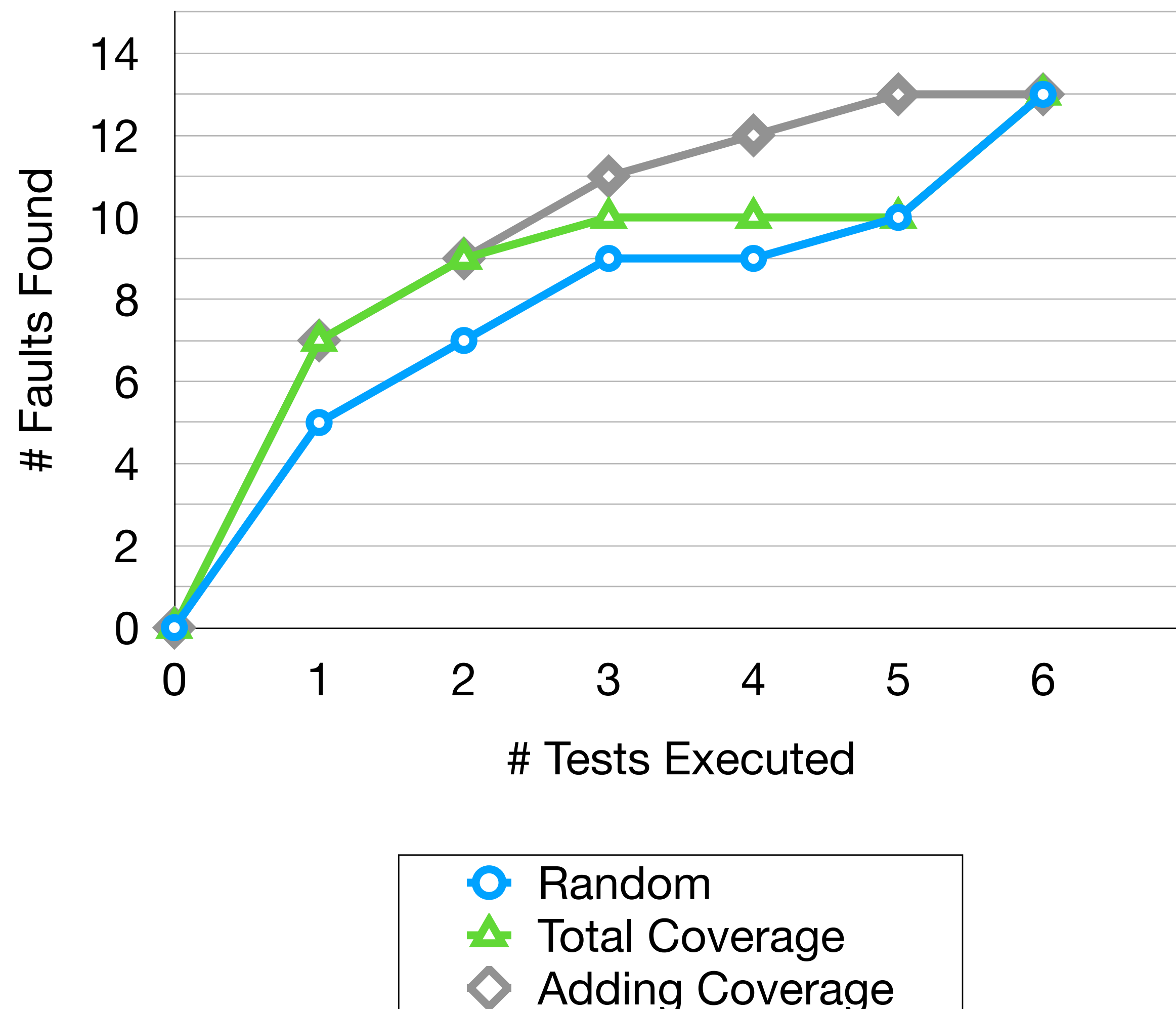
Using Fault-Detection Data



APFD: Average Percentage of Faults Detected

$$APFD = 1 - \frac{TF_1 + TF_2 + \dots + TF_m}{nm} + \frac{1}{2n}$$

Using Fault-Detection Data

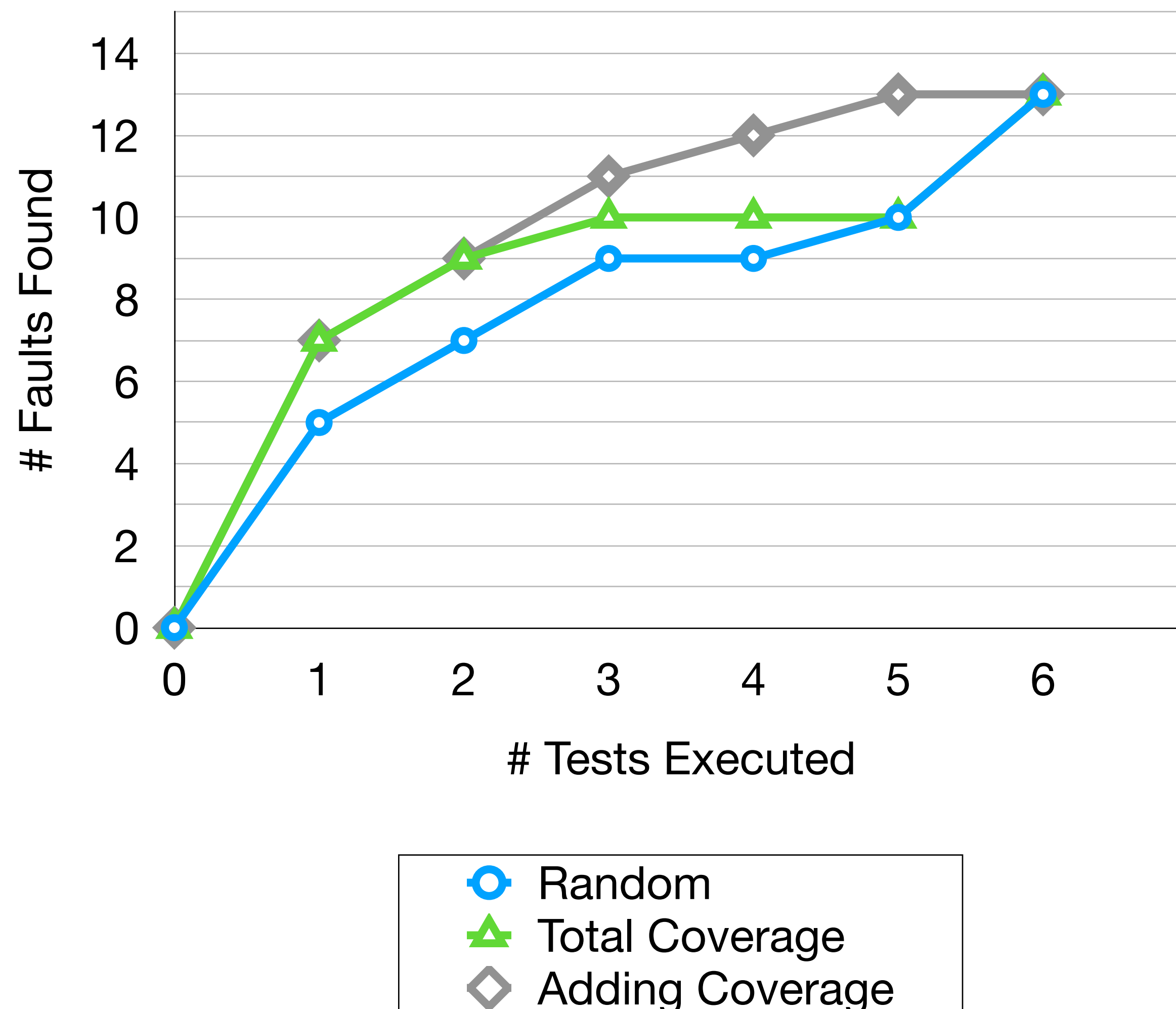


APFD: Average Percentage of Faults Detected

$$APFD = 1 - \frac{TF_1 + TF_2 + \dots + TF_m}{nm} + \frac{1}{2n}$$

Where

Using Fault-Detection Data



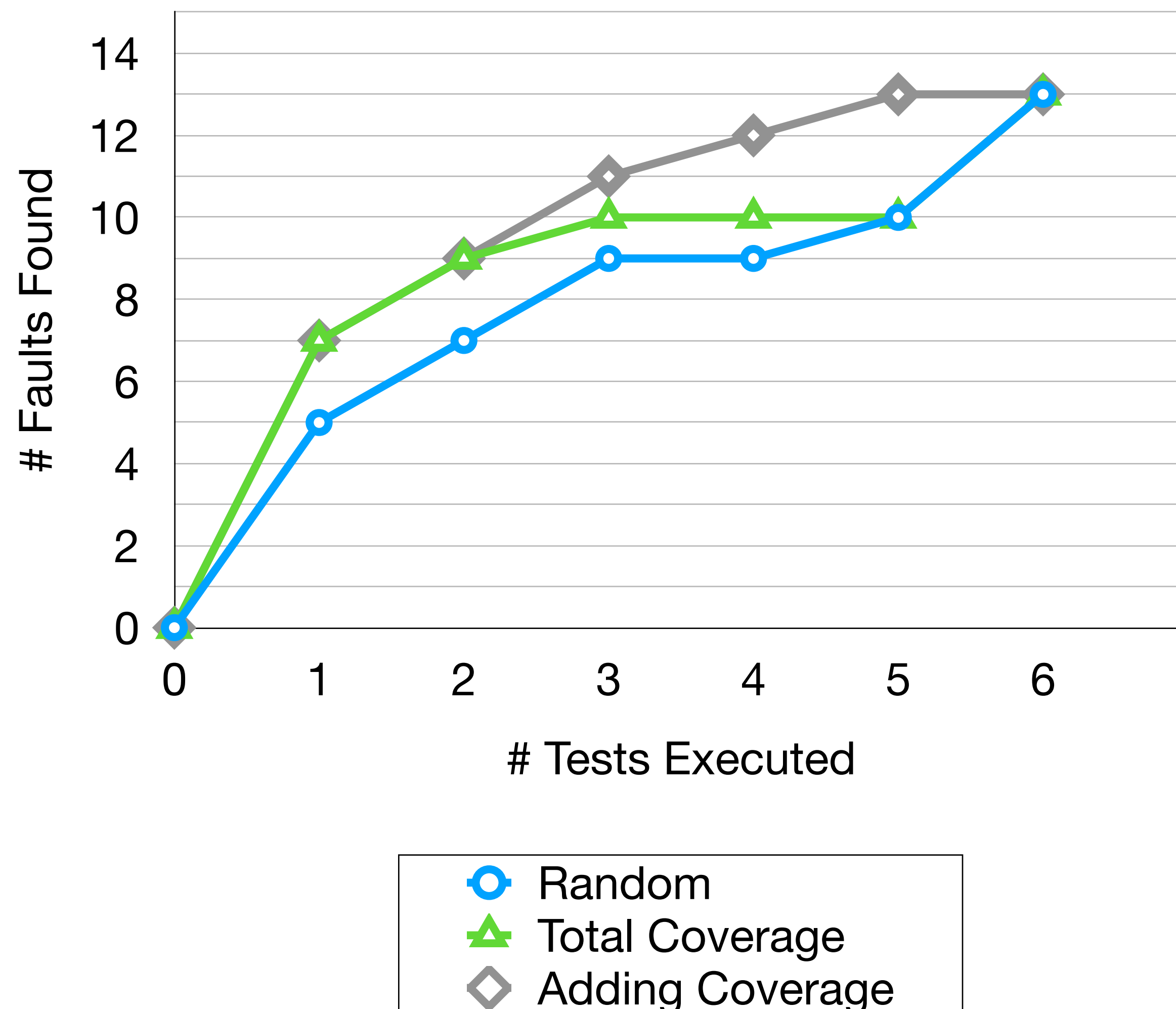
APFD: Average Percentage of Faults Detected

$$APFD = 1 - \frac{TF_1 + TF_2 + \dots + TF_m}{nm} + \frac{1}{2n}$$

Where

TF_i is the position of the first test that kills mutant i

Using Fault-Detection Data



APFD: Average Percentage of Faults Detected

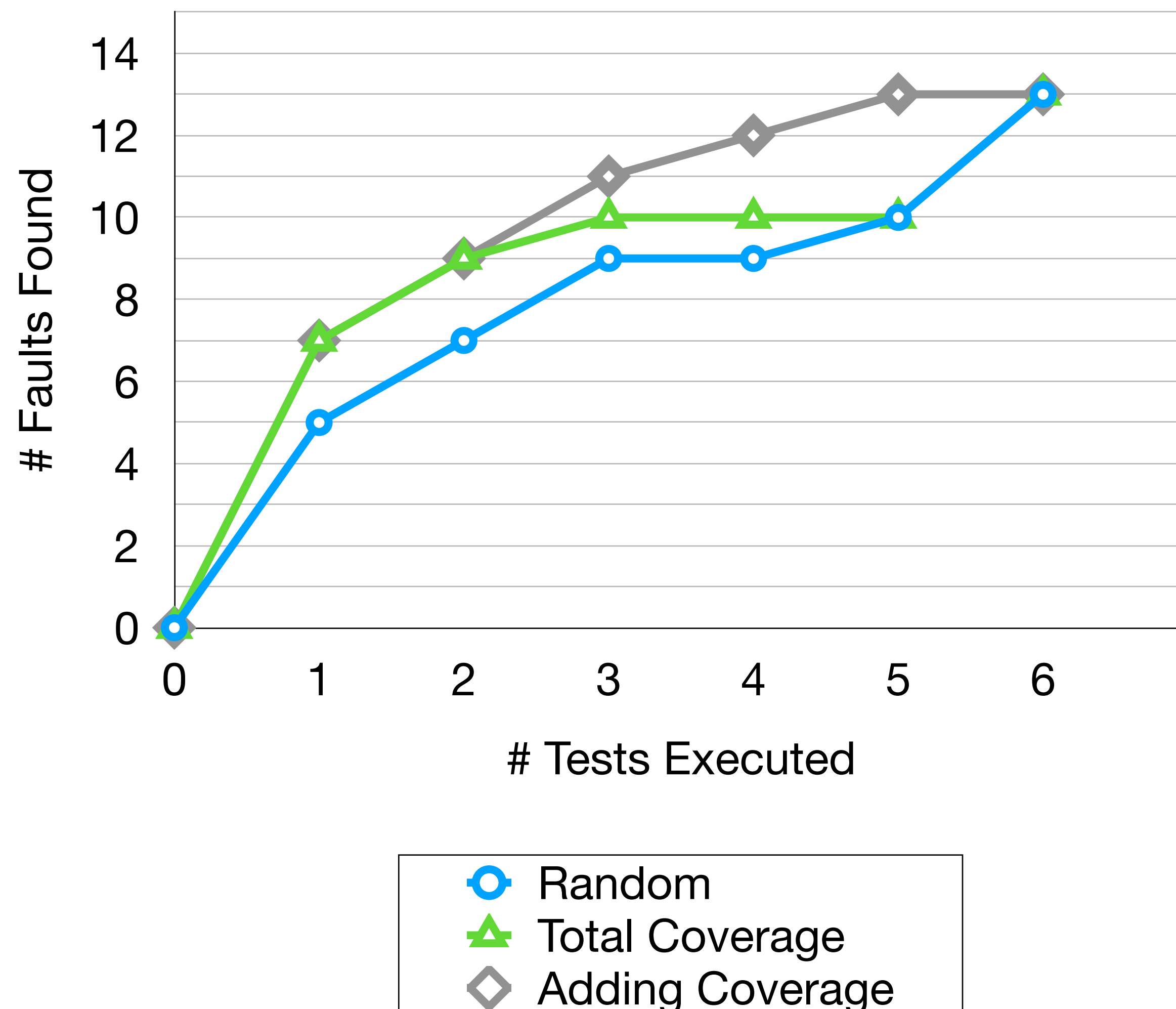
$$APFD = 1 - \frac{TF_1 + TF_2 + \dots + TF_m}{nm} + \frac{1}{2n}$$

Where

TF_i is the position of the first test that kills mutant i

m is the number of killed mutants

Using Fault-Detection Data



APFD: Average Percentage of Faults Detected

$$APFD = 1 - \frac{TF_1 + TF_2 + \dots + TF_m}{nm} + \frac{1}{2n}$$

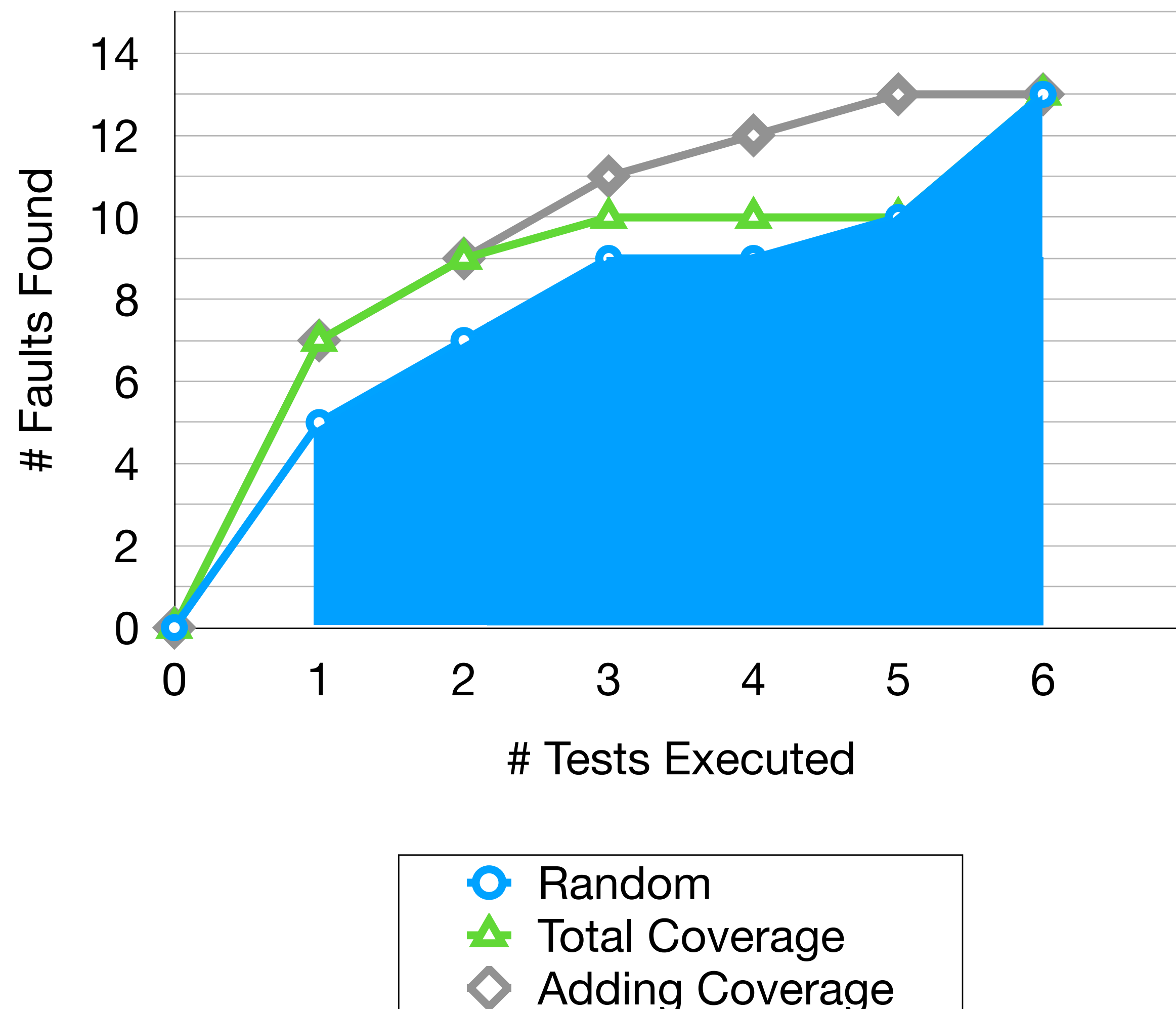
Where

TF_i is the position of the first test that kills mutant i

m is the number of killed mutants

n is the number of tests

Using Fault-Detection Data



APFD: Average Percentage of Faults Detected

$$APFD = 1 - \frac{TF_1 + TF_2 + \dots + TF_m}{nm} + \frac{1}{2n}$$

Where

TF_i is the position of the first test that kills mutant i

m is the number of killed mutants

n is the number of tests

Fitness Function for Coverage

Consider branch coverage (similar for other metrics)

How to assign a score/fitness to a particular order of tests?

Reward orders that are good for all prefixes

Standard Approach: use weighted average percentage of coverage over the sequence of tests

If we have n test cases, m branches, and TB_i is the number of the first test case that executes branch i , then the fitness is:

$$APC = 1 - \frac{TB_1 + \dots + TB_m}{nm} + \frac{1}{2n}$$

Optimisation

Similar to minimisation, we might have multiple objectives

- Different forms of coverage.

- Prioritising tests for historically faulty components.

- Prioritising tests for components based on fault prediction techniques.

Potential for Multi-Objective Optimisation

Test Selection

Test Selection

Minimisation aims to remove redundant tests

But is this the best idea in the long term?

A removed test may have been **useful to reveal a bug in the future**

Selection is an alternative to **minimisation**

It's ok to keep large test suites in the codebase

As long as which tests are run upon every change is decided carefully

Identification of affected code + Selection of tests to run

Intra-Procedural Test Selection

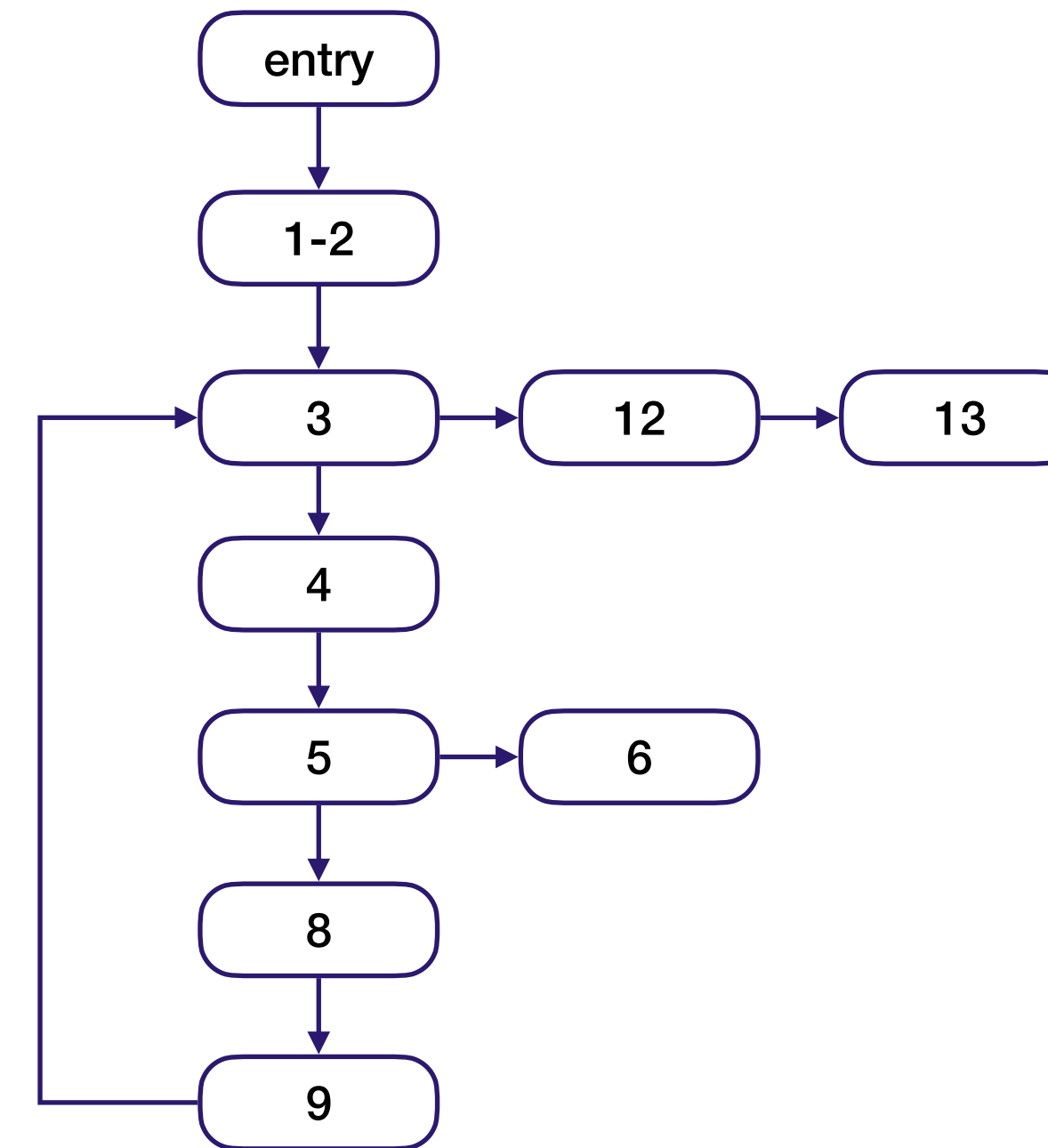
Identify **affected** code based on **parallel traversal** of the CFGs (control flow graphs) of the original and the modified versions

At each step: compare nodes for **lexicographical equivalence**

Difference found? Select tests traversing path from entry node to changed node, i.e., **modification-traversing tests**

An Average Example

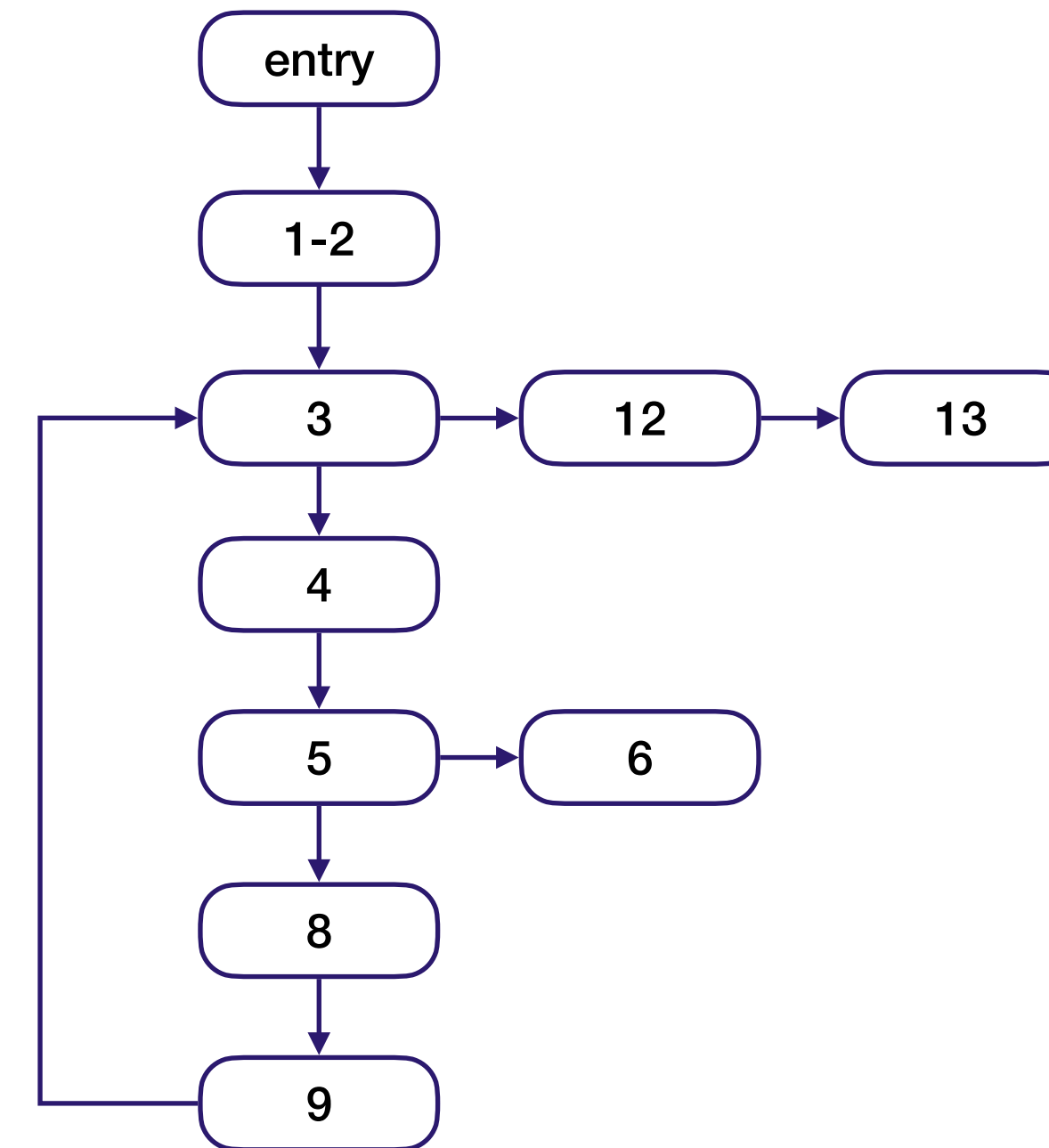
```
public static int avg(File inputFile) throws Exception {  
1  int count = 0; int sum = 0;  
2  Scanner in = new Scanner(new FileReader(inputFile));  
3  while(in.hasNext()) {  
4      String line = in.next();  
5      if (! StringUtils.isNumeric(line)) {  
6          throw new NumberFormatException();  
7      } else {  
8          sum += Integer.parseInt(line);  
9          count++;  
10     }  
11 }  
12 in.close();  
13 return count > 0 ? sum / count : 0;  
}
```



An Average Example

```
public static int avg(File inputFile) throws Exception {  
  1  int count = 0; int sum = 0;  
  2  Scanner in = new Scanner(new FileReader(inputFile));  
  3  while(in.hasNext()) {  
  4      String line = in.next();  
  5      if (! StringUtils.isNumeric(line)) {  
  6          throw new NumberFormatException();  
  7      } else {  
  8          sum += Integer.parseInt(line);  
  9          count++;  
 10      }  
 11  }  
 12  in.close();  
 13  return count > 0 ? sum / count : 0;  
}
```

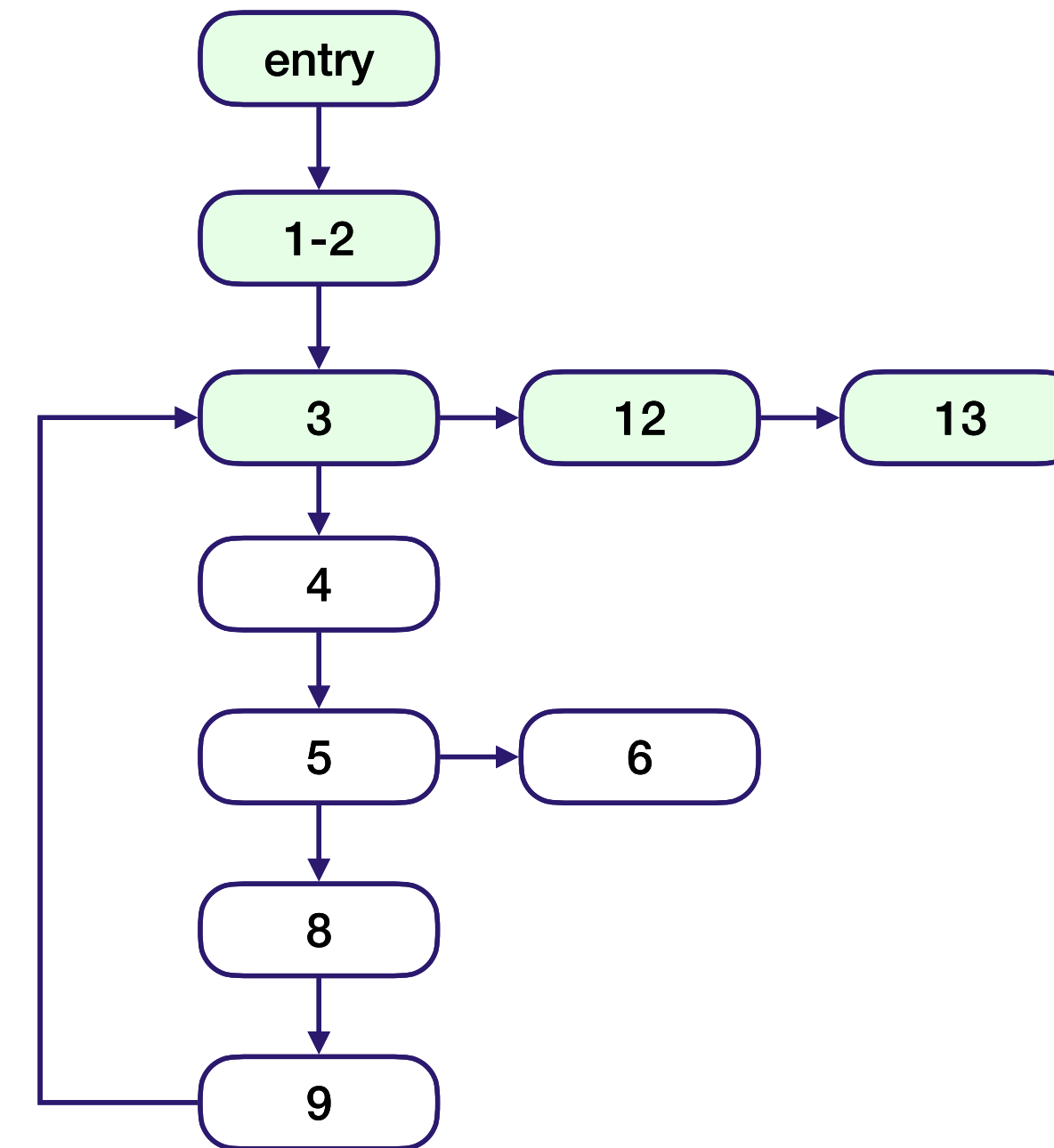
```
@Test  
public void t1() throws Exception {  
    assertEquals(0, TestSelection.avg(empty));  
}
```



An Average Example

```
public static int avg(File inputFile) throws Exception {  
  1  int count = 0; int sum = 0;  
  2  Scanner in = new Scanner(new FileReader(inputFile));  
  3  while(in.hasNext()) {  
  4      String line = in.next();  
  5      if (! StringUtils.isNumeric(line)) {  
  6          throw new NumberFormatException();  
  7      } else {  
  8          sum += Integer.parseInt(line);  
  9          count++;  
 10      }  
 11  }  
 12  in.close();  
 13  return count > 0 ? sum / count : 0;  
}
```

```
@Test  
public void t1() throws Exception {  
    assertEquals(0, TestSelection.avg(empty));  
}
```

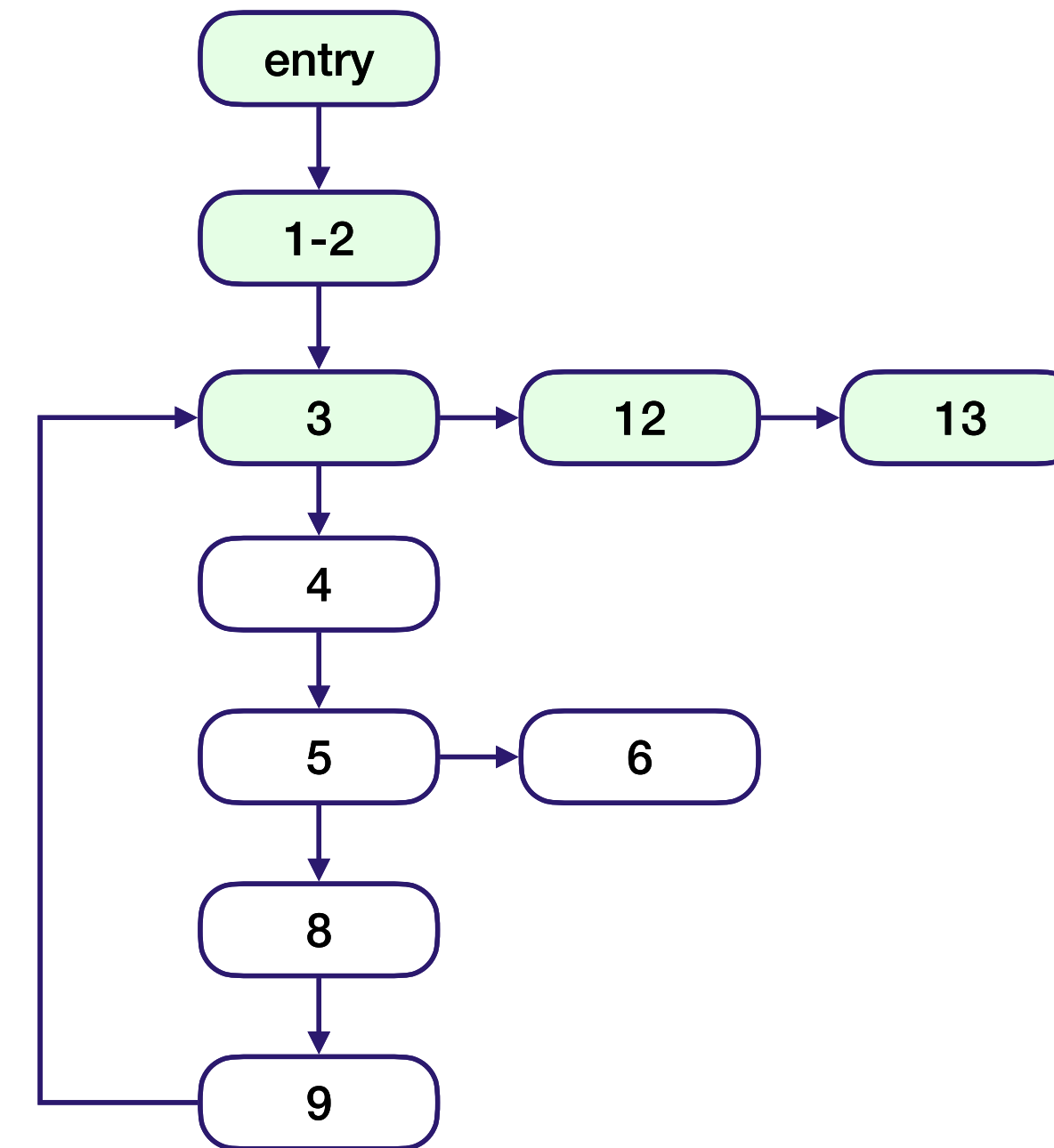


An Average Example

```
public static int avg(File inputFile) throws Exception {  
  1  int count = 0; int sum = 0;  
  2  Scanner in = new Scanner(new FileReader(inputFile));  
  3  while(in.hasNext()) {  
  4      String line = in.next();  
  5      if (! StringUtils.isNumeric(line)) {  
  6          throw new NumberFormatException();  
  7      } else {  
  8          sum += Integer.parseInt(line);  
  9          count++;  
 10      }  
 11  }  
 12  in.close();  
 13  return count > 0 ? sum / count : 0;  
}
```

```
@Test  
public void t1() throws Exception {  
    assertEquals(0, TestSelection.avg(empty));  
}
```

```
@Test  
public void t2() throws Exception {  
    assertThrows(NumberFormatException.class, () -> {  
        TestSelection.avg(nonNumeric);  
    });  
}
```

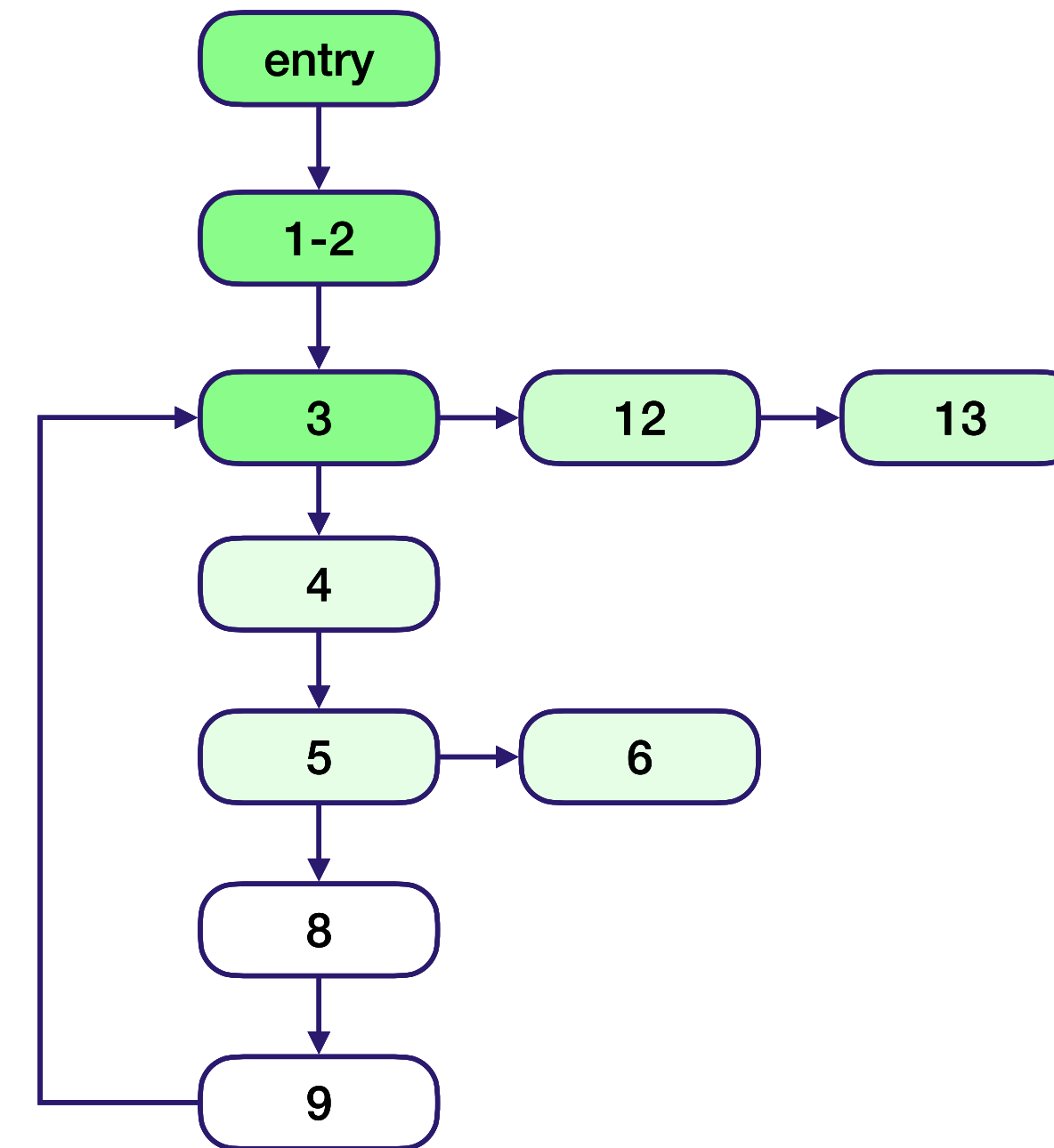


An Average Example

```
public static int avg(File inputFile) throws Exception {  
  1  int count = 0; int sum = 0;  
  2  Scanner in = new Scanner(new FileReader(inputFile));  
  3  while(in.hasNext()) {  
  4      String line = in.next();  
  5      if (! StringUtils.isNumeric(line)) {  
  6          throw new NumberFormatException();  
  7      } else {  
  8          sum += Integer.parseInt(line);  
  9          count++;  
 10      }  
 11  }  
 12  in.close();  
 13  return count > 0 ? sum / count : 0;  
}
```

```
@Test  
public void t1() throws Exception {  
    assertEquals(0, TestSelection.avg(empty));  
}
```

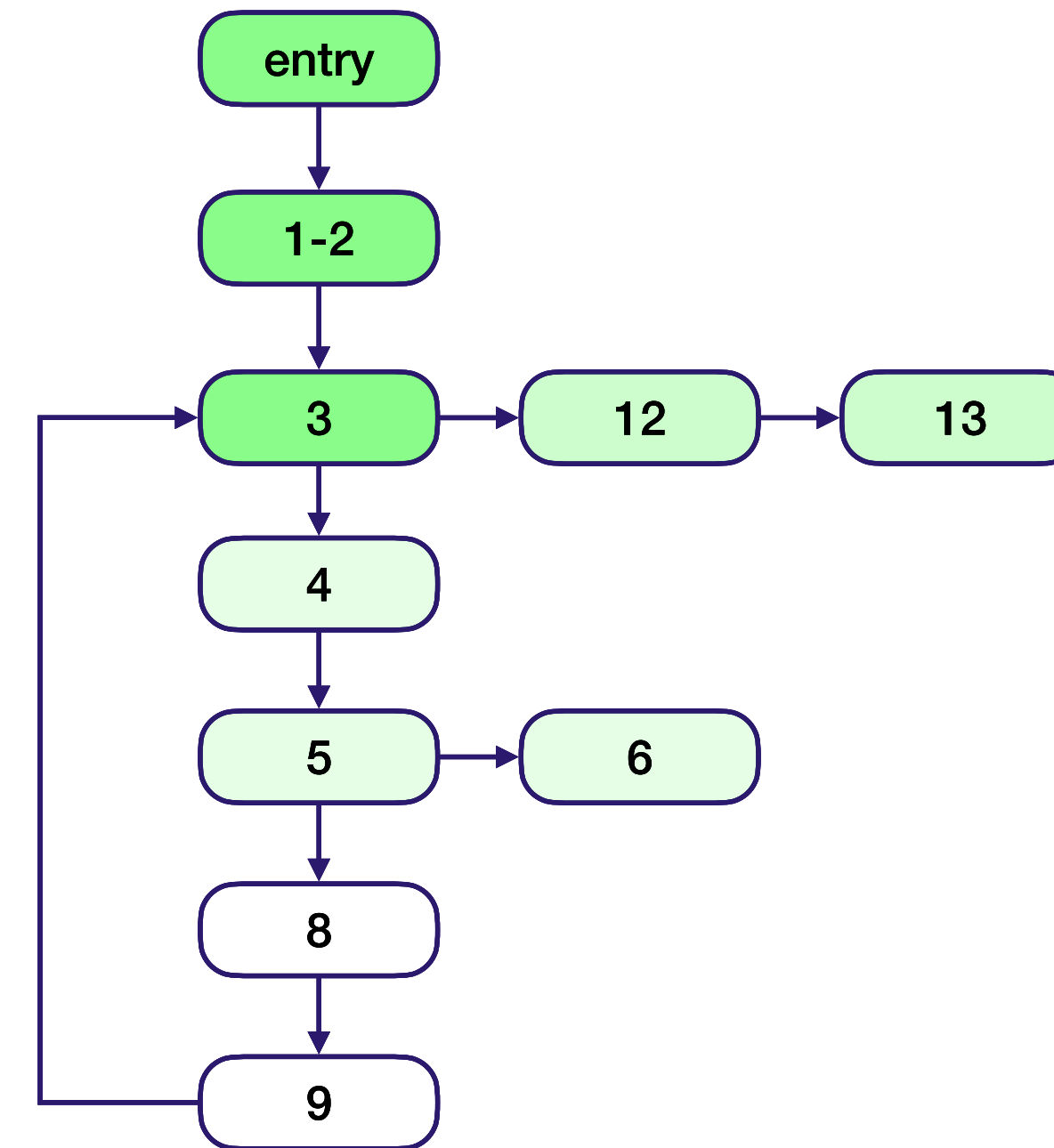
```
@Test  
public void t2() throws Exception {  
    assertThrows(NumberFormatException.class, () -> {  
        TestSelection.avg(nonNumeric);  
    });  
}
```



An Average Example

```
public static int avg(File inputFile) throws Exception {  
  1  int count = 0; int sum = 0;  
  2  Scanner in = new Scanner(new FileReader(inputFile));  
  3  while(in.hasNext()) {  
  4      String line = in.next();  
  5      if (! StringUtils.isNumeric(line)) {  
  6          throw new NumberFormatException();  
  7      } else {  
  8          sum += Integer.parseInt(line);  
  9          count++;  
 10      }  
 11  }  
 12  in.close();  
 13  return count > 0 ? sum / count : 0;  
}
```

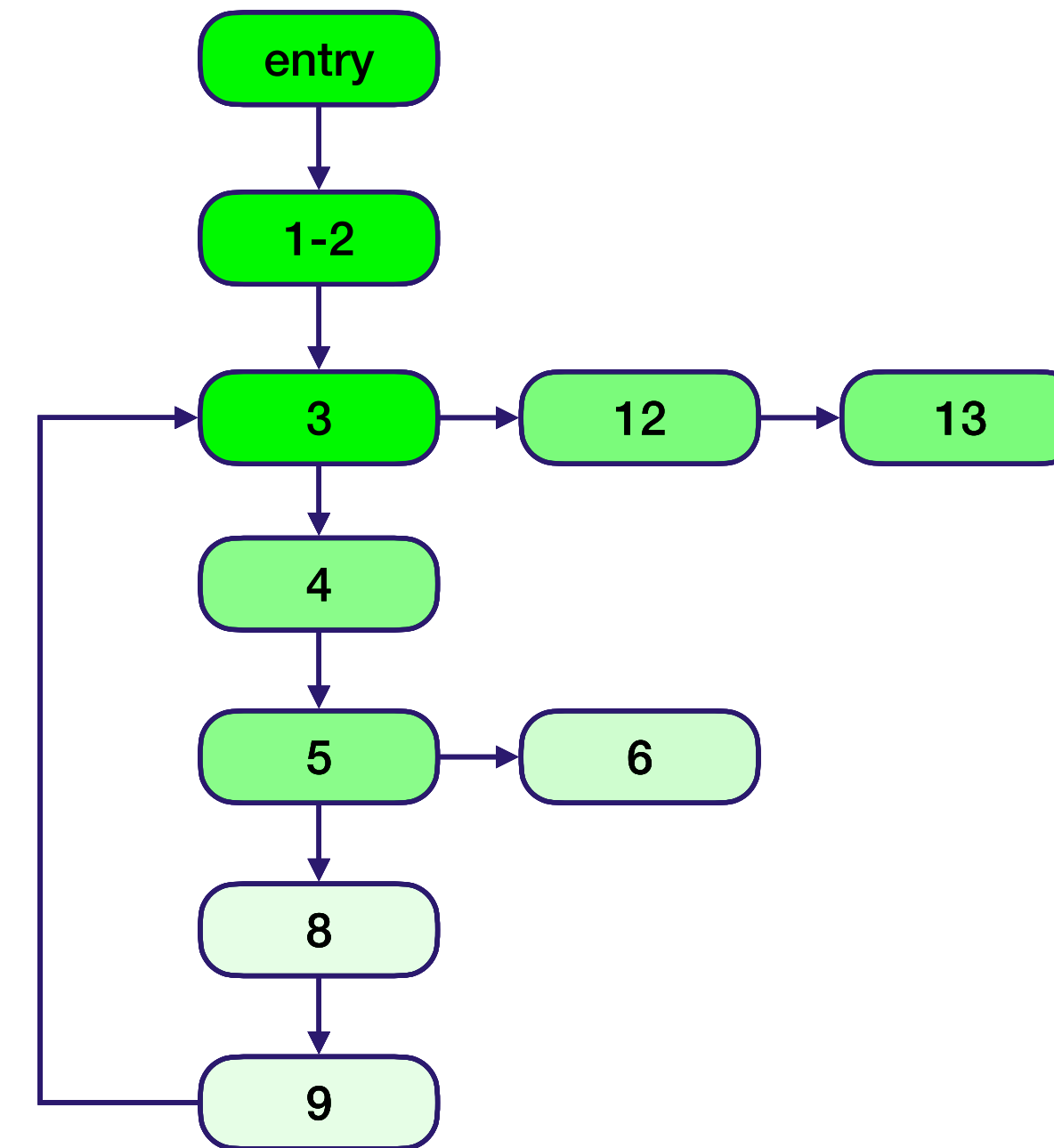
```
@Test  
public void t1() throws Exception {  
    assertEquals(0, TestSelection.avg(empty));  
}  
  
@Test  
public void t2() throws Exception {  
    assertThrows(NumberFormatException.class, () -> {  
        TestSelection.avg(nonNumeric);  
    });  
}  
  
@Test  
public void t3() throws Exception {  
    assertEquals(2, TestSelection.avg(numbers123));  
}
```



An Average Example

```
public static int avg(File inputFile) throws Exception {  
  1  int count = 0; int sum = 0;  
  2  Scanner in = new Scanner(new FileReader(inputFile));  
  3  while(in.hasNext()) {  
  4      String line = in.next();  
  5      if (! StringUtils.isNumeric(line)) {  
  6          throw new NumberFormatException();  
  7      } else {  
  8          sum += Integer.parseInt(line);  
  9          count++;  
 10      }  
 11  }  
 12  in.close();  
 13  return count > 0 ? sum / count : 0;  
}
```

```
@Test  
public void t1() throws Exception {  
    assertEquals(0, TestSelection.avg(empty));  
}  
  
@Test  
public void t2() throws Exception {  
    assertThrows(NumberFormatException.class, () -> {  
        TestSelection.avg(nonNumeric);  
    });  
}  
  
@Test  
public void t3() throws Exception {  
    assertEquals(2, TestSelection.avg(numbers123));  
}
```



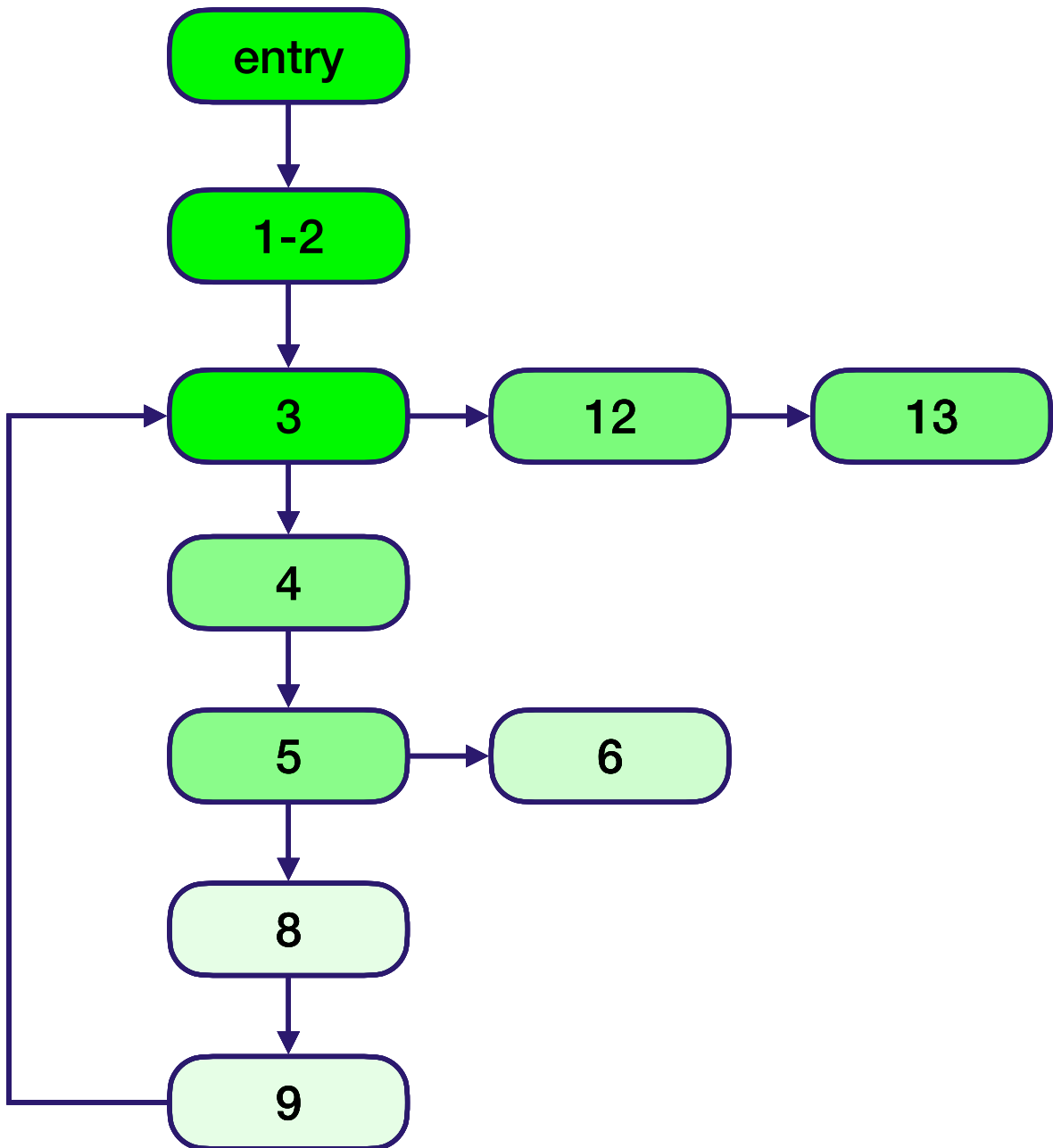
An Average Example

```
public static int avg(File inputFile) throws Exception {
1  int count = 0; int sum = 0;
2  Scanner in = new Scanner(new FileReader(inputFile));
3  while(in.hasNext()) {
4      String line = in.next();
5      if (! StringUtils.isNumeric(line)) {
6          throw new NumberFormatException();
7      } else {
8          sum += Integer.parseInt(line);
9          count++;
10     }
11 }
12 in.close();
13 return count > 0 ? sum / count : 0;
}
```

```
@Test
public void t1() throws Exception {
    assertEquals(0, TestSelection.avg(empty));
}

@Test
public void t2() throws Exception {
    assertThrows(NumberFormatException.class, () -> {
        TestSelection.avg(nonNumeric);
    });
}

@Test
public void t3() throws Exception {
    assertEquals(2, TestSelection.avg(numbers123));
}
```



Test	Edges Traversed
------	-----------------

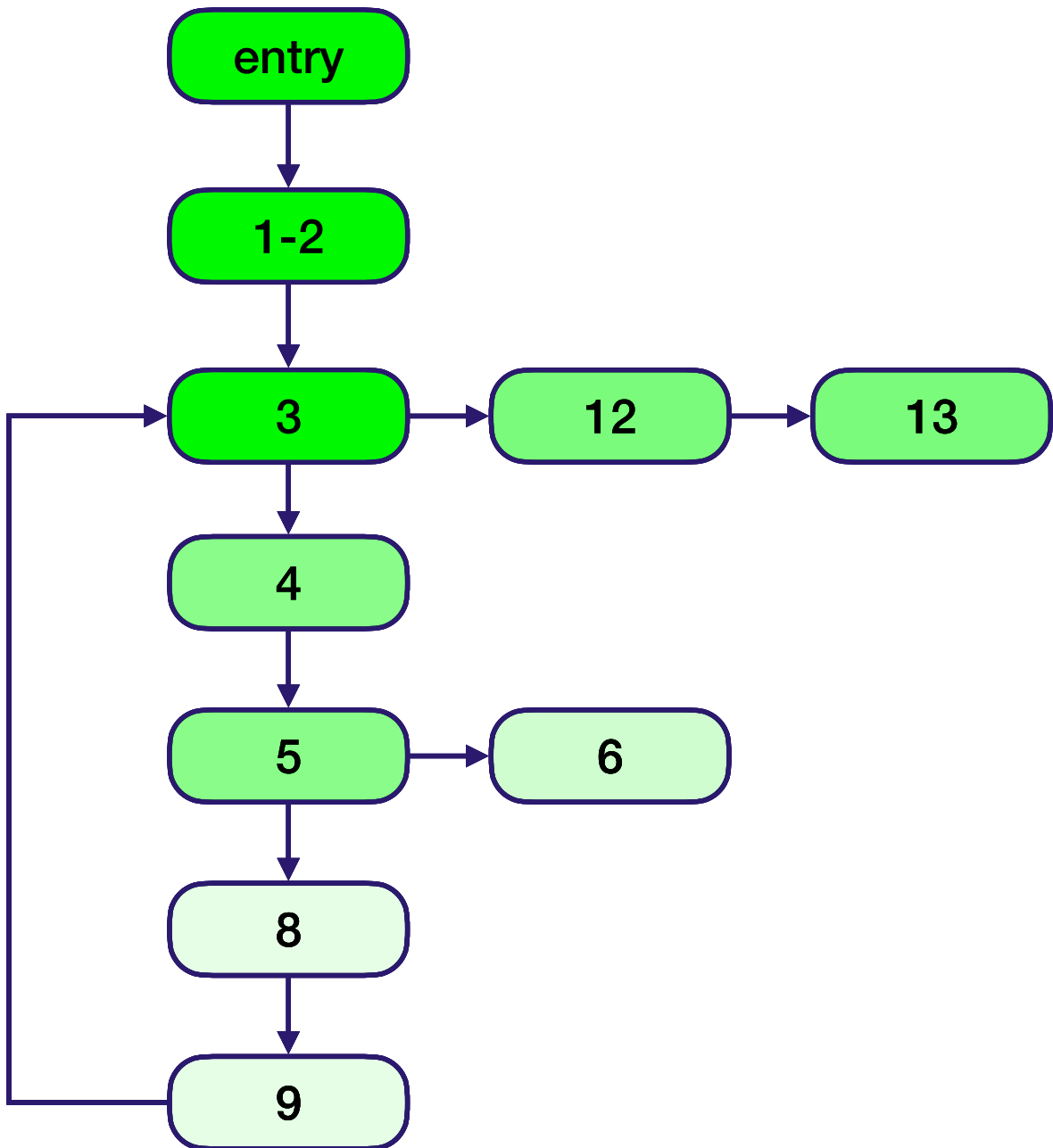
An Average Example

```
public static int avg(File inputFile) throws Exception {
1  int count = 0; int sum = 0;
2  Scanner in = new Scanner(new FileReader(inputFile));
3  while(in.hasNext()) {
4      String line = in.next();
5      if (! StringUtils.isNumeric(line)) {
6          throw new NumberFormatException();
7      } else {
8          sum += Integer.parseInt(line);
9          count++;
10     }
11 }
12 in.close();
13 return count > 0 ? sum / count : 0;
}
```

```
@Test
public void t1() throws Exception {
    assertEquals(0, TestSelection.avg(empty));
}

@Test
public void t2() throws Exception {
    assertThrows(NumberFormatException.class, () -> {
        TestSelection.avg(nonNumeric);
    });
}

@Test
public void t3() throws Exception {
    assertEquals(2, TestSelection.avg(numbers123));
}
```



Test	Edges Traversed
t1	(entry, 1-2), (1-2, 3), (3, 12), (12, 13)

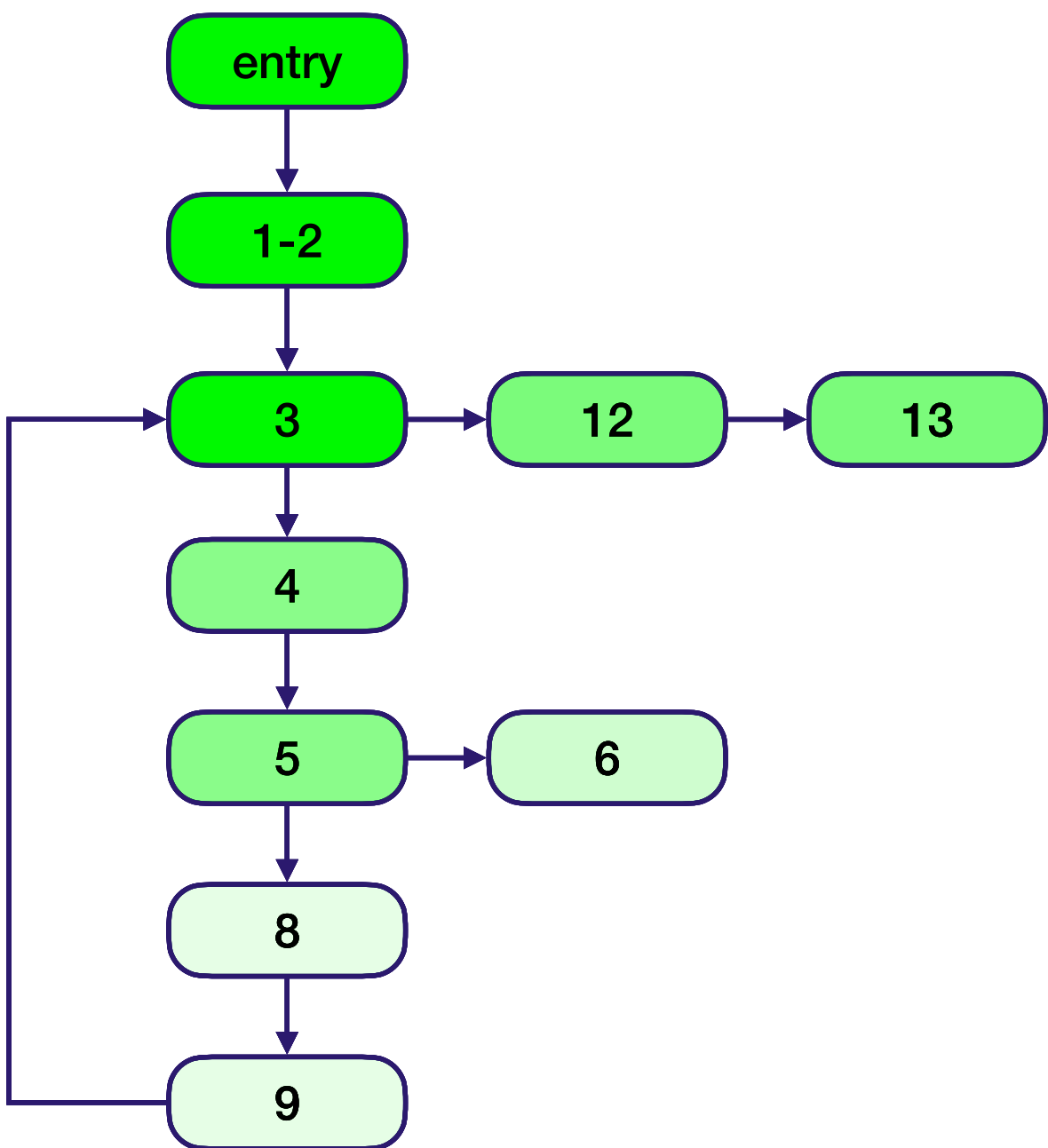
An Average Example

```
public static int avg(File inputFile) throws Exception {
1  int count = 0; int sum = 0;
2  Scanner in = new Scanner(new FileReader(inputFile));
3  while(in.hasNext()) {
4      String line = in.next();
5      if (! StringUtils.isNumeric(line)) {
6          throw new NumberFormatException();
7      } else {
8          sum += Integer.parseInt(line);
9          count++;
10     }
11 }
12 in.close();
13 return count > 0 ? sum / count : 0;
}
```

```
@Test
public void t1() throws Exception {
    assertEquals(0, TestSelection.avg(empty));
}

@Test
public void t2() throws Exception {
    assertThrows(NumberFormatException.class, () -> {
        TestSelection.avg(nonNumeric);
    });
}

@Test
public void t3() throws Exception {
    assertEquals(2, TestSelection.avg(numbers123));
}
```



Test	Edges Traversed
t1	(entry, 1-2), (1-2, 3), (3, 12), (12, 13)
t2	(entry, 1-2), (1-2, 3), (3, 4), (4, 5), (5, 6)

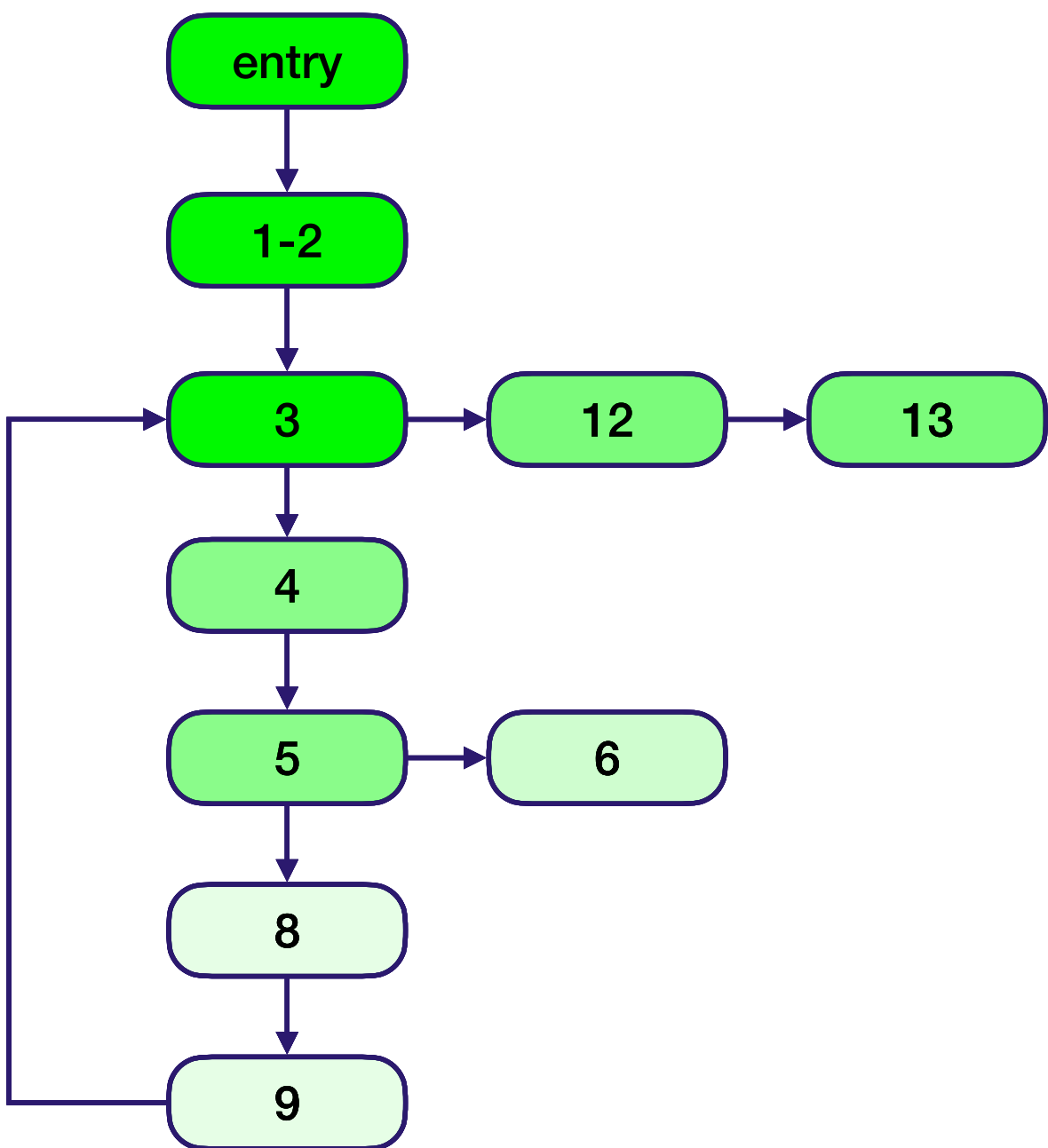
An Average Example

```
public static int avg(File inputFile) throws Exception {
1  int count = 0; int sum = 0;
2  Scanner in = new Scanner(new FileReader(inputFile));
3  while(in.hasNext()) {
4      String line = in.next();
5      if (! StringUtils.isNumeric(line)) {
6          throw new NumberFormatException();
7      } else {
8          sum += Integer.parseInt(line);
9          count++;
10     }
11 }
12 in.close();
13 return count > 0 ? sum / count : 0;
}
```

```
@Test
public void t1() throws Exception {
    assertEquals(0, TestSelection.avg(empty));
}

@Test
public void t2() throws Exception {
    assertThrows(NumberFormatException.class, () -> {
        TestSelection.avg(nonNumeric);
    });
}

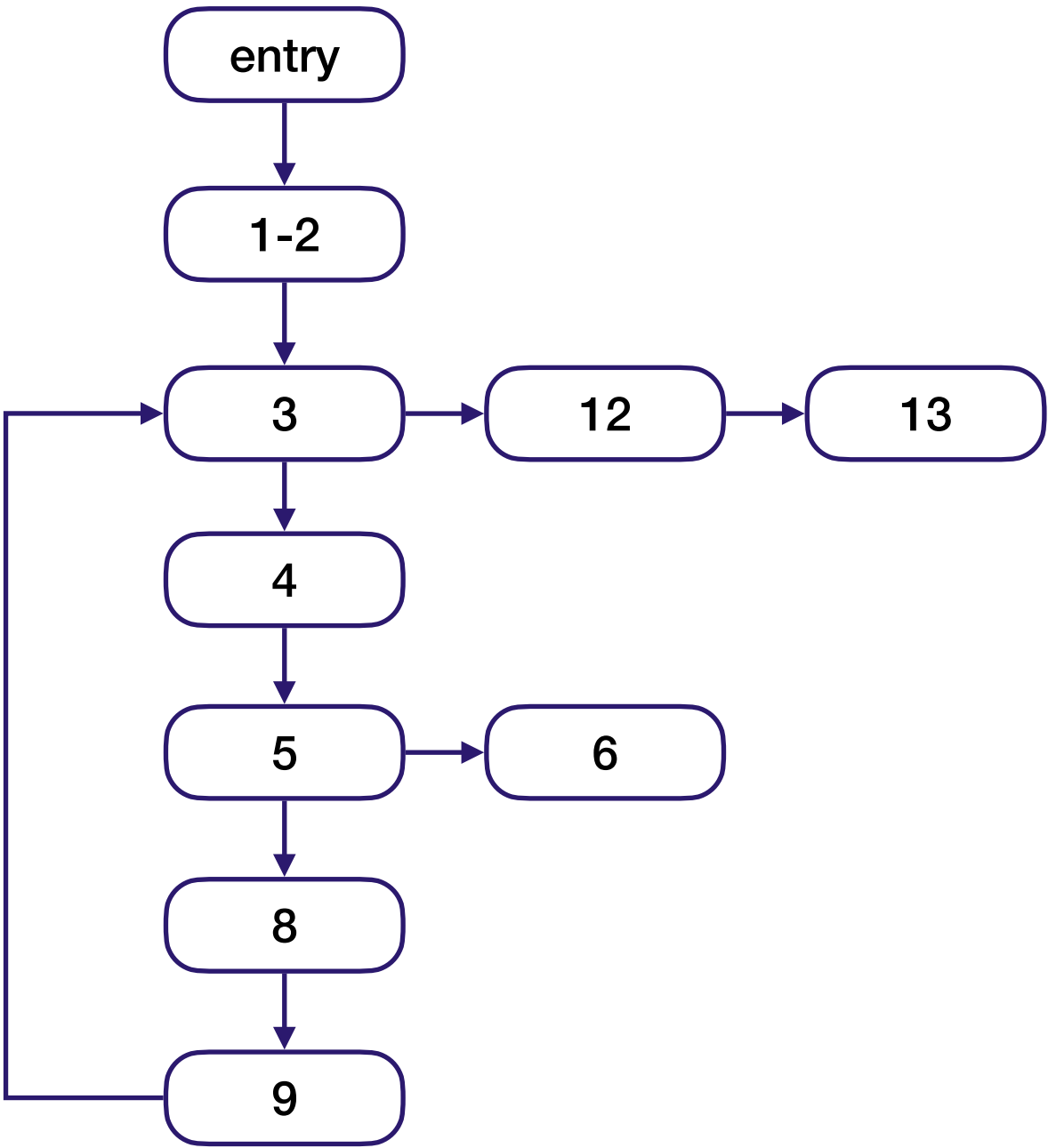
@Test
public void t3() throws Exception {
    assertEquals(2, TestSelection.avg(numbers123));
}
```



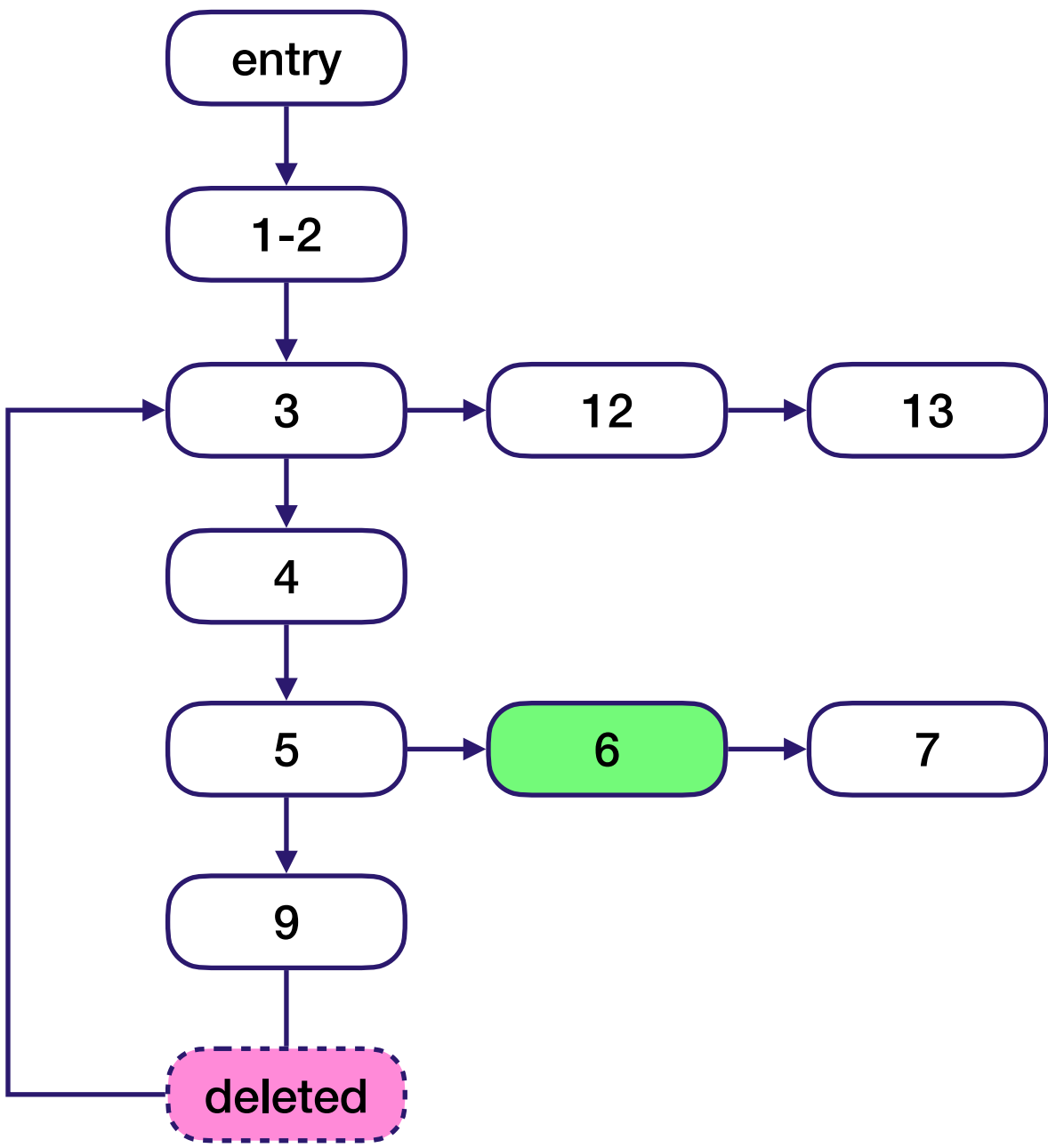
Test	Edges Traversed
t1	(entry, 1-2), (1-2, 3), (3, 12), (12, 13)
t2	(entry, 1-2), (1-2, 3), (3, 4), (4, 5), (5, 6)
t3	(entry, 1-2), (1-2, 3), (3, 4), (4, 5), (5, 8), (8, 9), (9, 3), (3, 12), (12, 13)

An Average Example

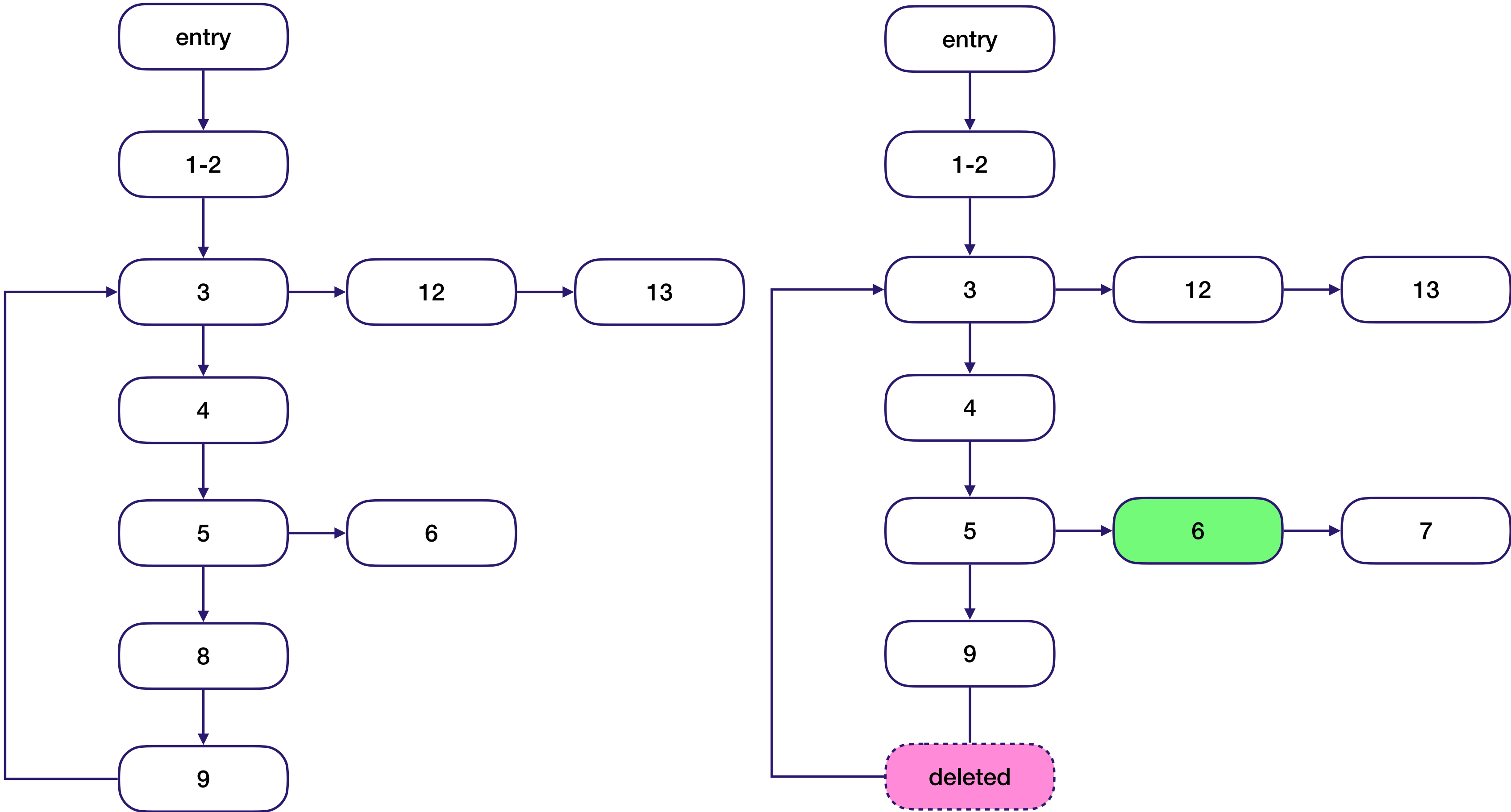
```
public static int avg(File inputFile) throws Exception {
1  int count = 0; int sum = 0;
2  Scanner in = new Scanner(new FileReader(inputFile));
3  while(in.hasNext()) {
4      String line = in.next();
5      if (! StringUtils.isNumeric(line)) {
6          throw new Exception();
7      } else {
8          sum += Integer.parseInt(line);
9          count++;
10     }
11 }
12 in.close();
13 return count > 0 ? sum / count : 0;
}
```



```
public static int avg(File inputFile) throws Exception {
1  int count = 0; int sum = 0;
2  Scanner in = new Scanner(new FileReader(inputFile));
3  while(in.hasNext()) {
4      String line = in.next();
5      if (! StringUtils.isNumeric(line)) {
6          System.err.println("Bad input");
7          throw new Exception();
8      } else {
9          sum += Integer.parseInt(line);
10     }
11 }
12 in.close();
13 return count > 0 ? sum / count : 0;
}
```



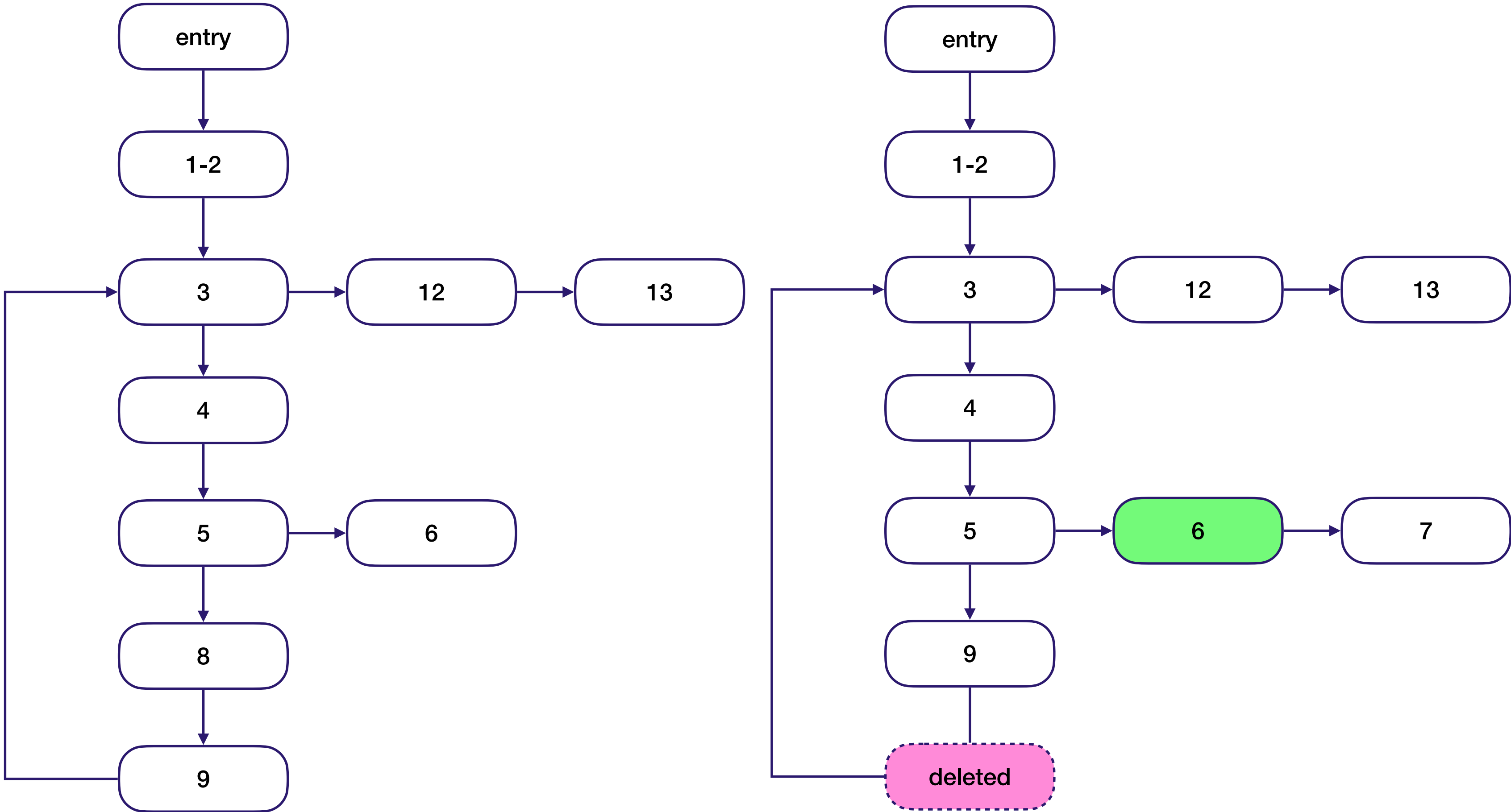
An Average Example



Test	Edges Traversed
t1	(entry, 1-2), (1-2, 3), (3, 12), (12, 13)
t2	(entry, 1-2), (1-2, 3), (3, 4), (4, 5), (5, 6)
t3	(entry, 1-2), (1-2, 3), (3, 4), (4, 5), (5, 8), (8, 9), (9, 3), (3, 12), (12, 13)

An Average Example

Start with **selection** = \emptyset

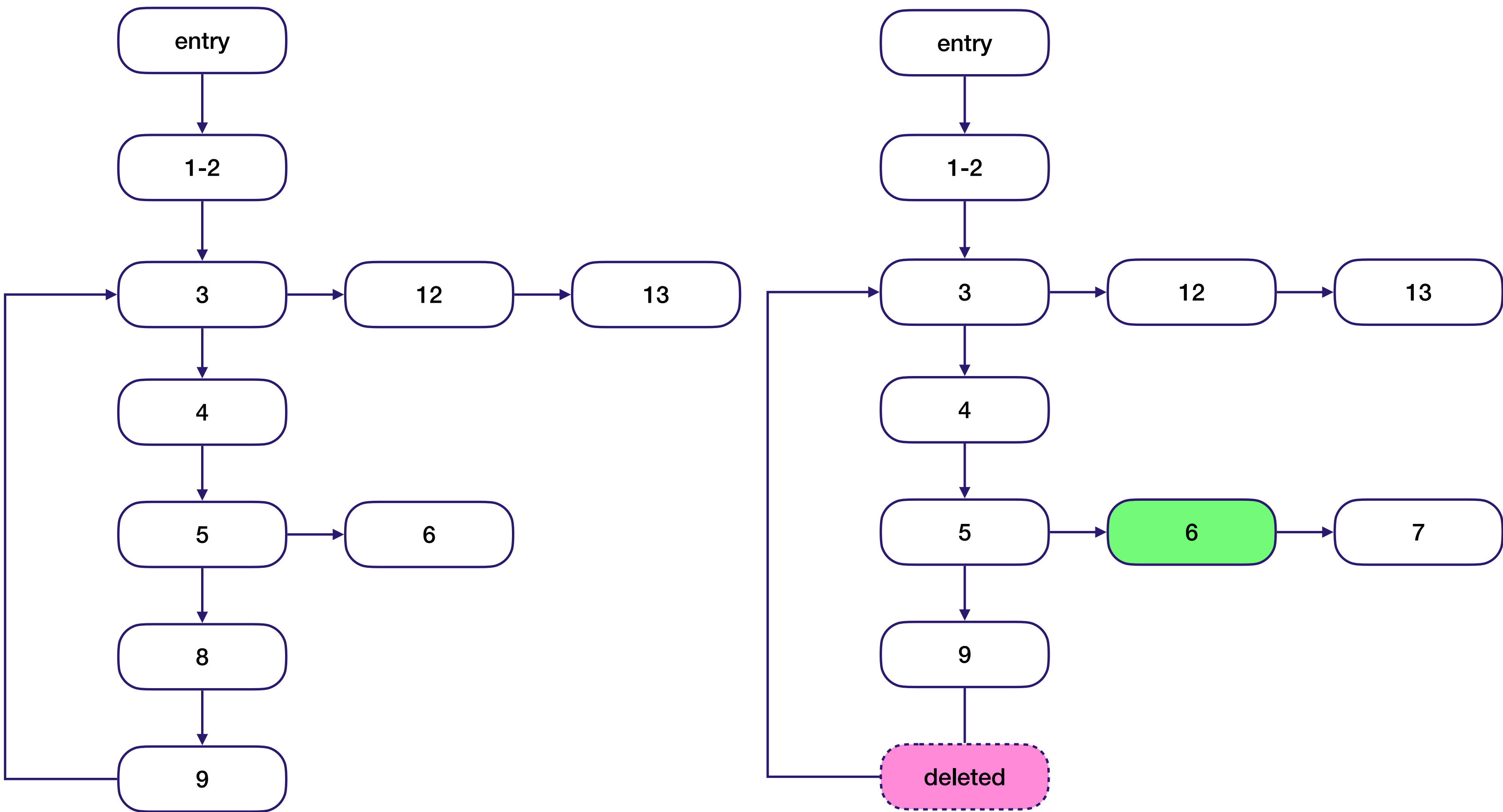


Test	Edges Traversed
t1	(entry, 1-2), (1-2, 3), (3, 12), (12, 13)
t2	(entry, 1-2), (1-2, 3), (3, 4), (4, 5), (5, 6)
t3	(entry, 1-2), (1-2, 3), (3, 4), (4, 5), (5, 8), (8, 9), (9, 3), (3, 12), (12, 13)

An Average Example

Start with **selection** = \emptyset

Traverse CFGs in parallel



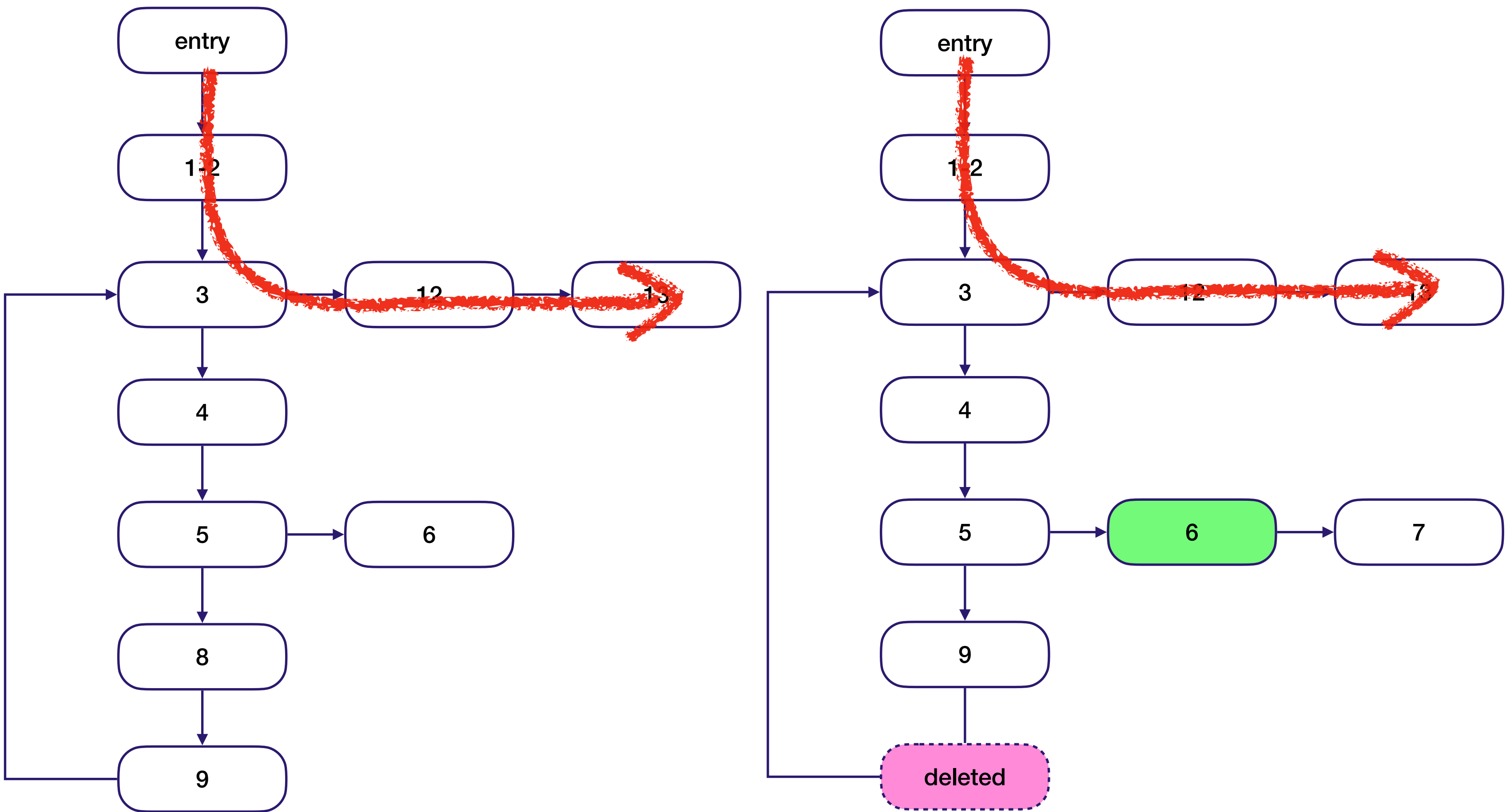
Test	Edges Traversed
t1	(entry, 1-2), (1-2, 3), (3, 12), (12, 13)
t2	(entry, 1-2), (1-2, 3), (3, 4), (4, 5), (5, 6)
t3	(entry, 1-2), (1-2, 3), (3, 4), (4, 5), (5, 8), (8, 9), (9, 3), (3, 12), (12, 13)

An Average Example

Start with **selection** = \emptyset

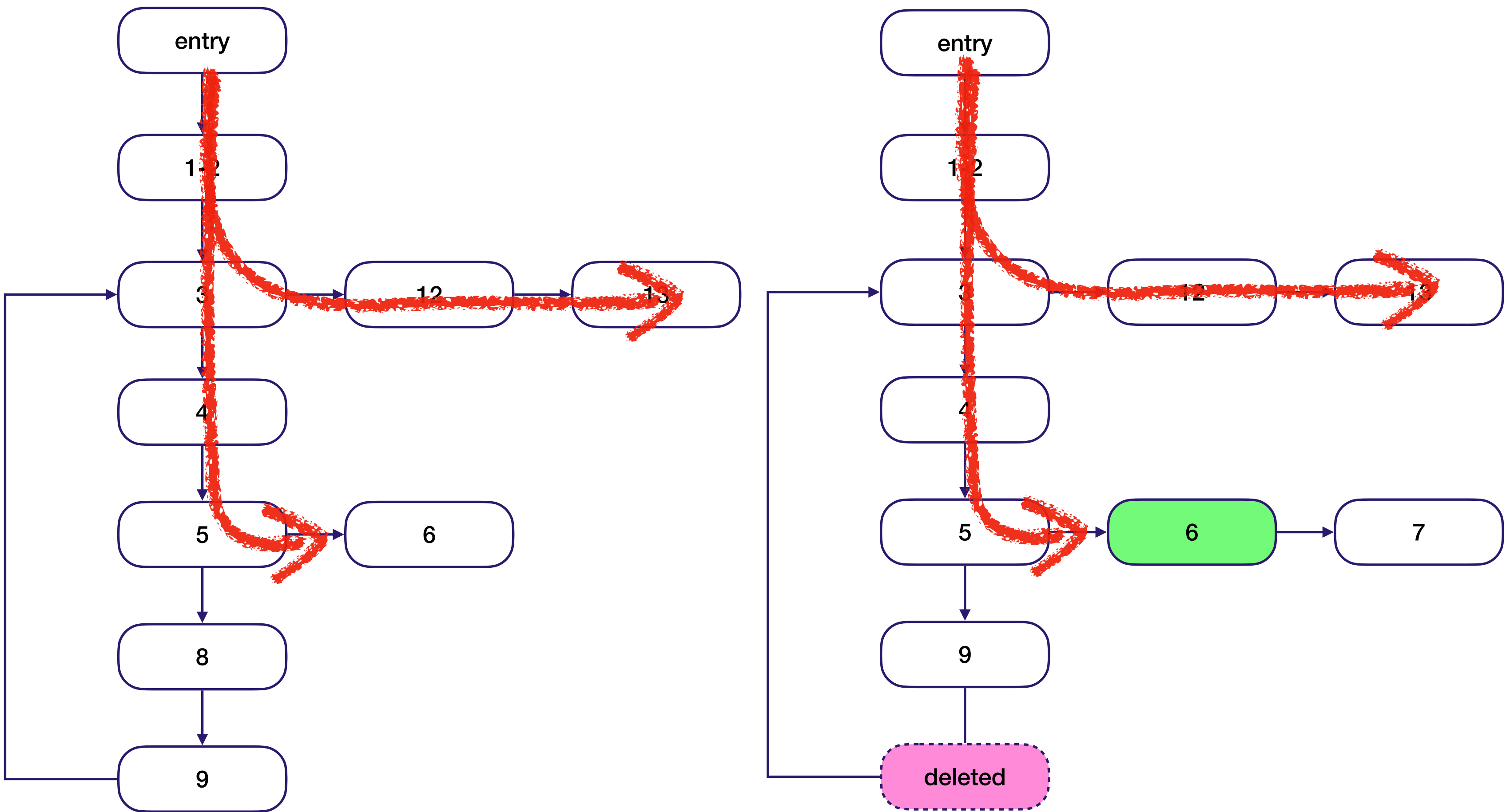
Traverse CFGs in parallel

To spot differences



Test	Edges Traversed
t1	(entry, 1-2), (1-2, 3), (3, 12), (12, 13)
t2	(entry, 1-2), (1-2, 3), (3, 4), (4, 5), (5, 6)
t3	(entry, 1-2), (1-2, 3), (3, 4), (4, 5), (5, 8), (8, 9), (9, 3), (3, 12), (12, 13)

An Average Example



Start with **selection** = \emptyset

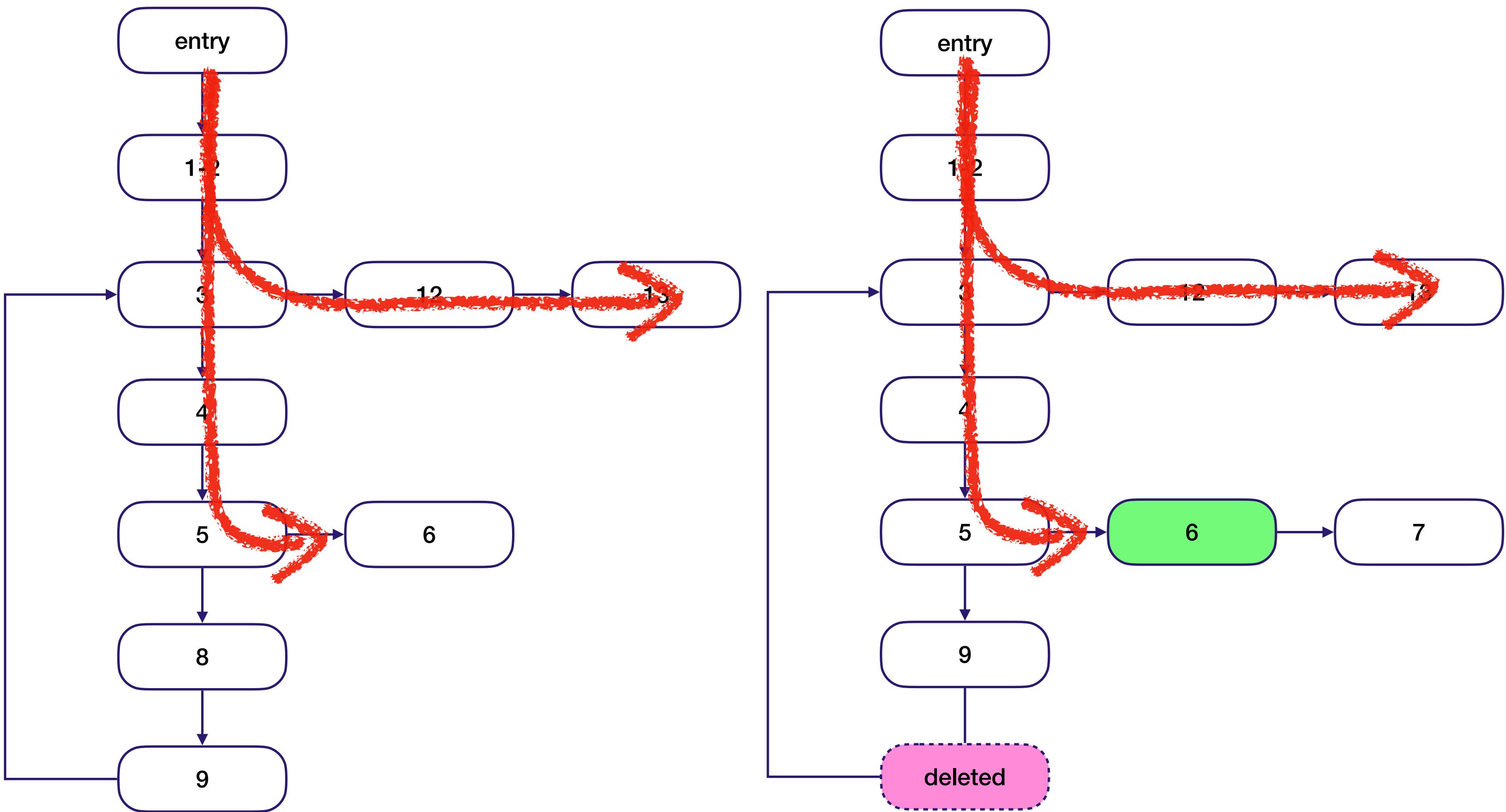
Traverse CFGs in parallel

To spot differences

Once difference is found

Test	Edges Traversed
t1	(entry, 1-2), (1-2, 3), (3, 12), (12, 13)
t2	(entry, 1-2), (1-2, 3), (3, 4), (4, 5), (5, 6)
t3	(entry, 1-2), (1-2, 3), (3, 4), (4, 5), (5, 8), (8, 9), (9, 3), (3, 12), (12, 13)

An Average Example



Start with **selection** = \emptyset

Traverse CFGs in parallel

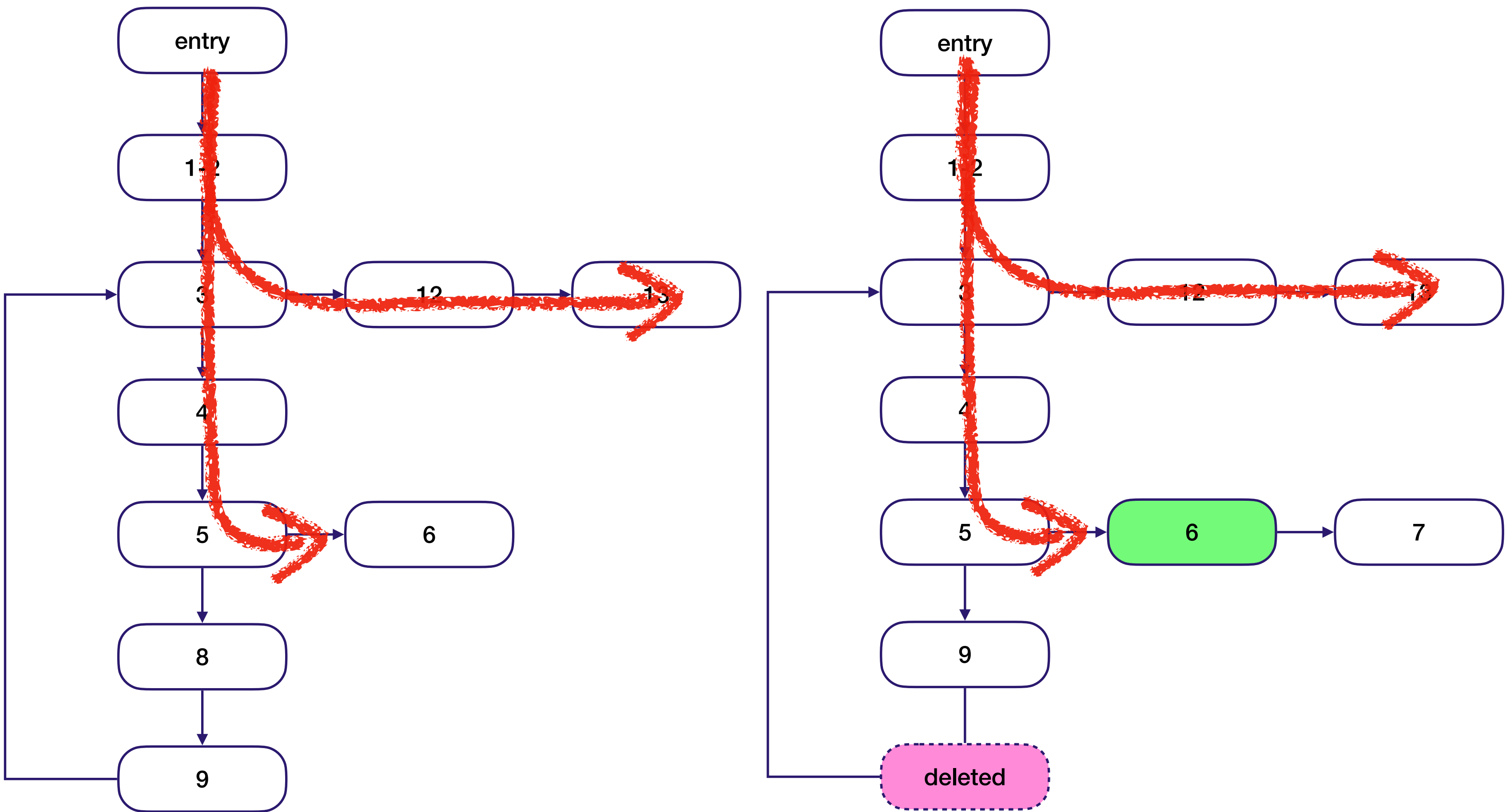
To spot differences

Once difference is found

Add test to selection

Test	Edges Traversed
t1	(entry, 1-2), (1-2, 3), (3, 12), (12, 13)
t2	(entry, 1-2), (1-2, 3), (3, 4), (4, 5), (5, 6)
t3	(entry, 1-2), (1-2, 3), (3, 4), (4, 5), (5, 8), (8, 9), (9, 3), (3, 12), (12, 13)

An Average Example



Start with **selection** = \emptyset

Traverse CFGs in parallel

To spot differences

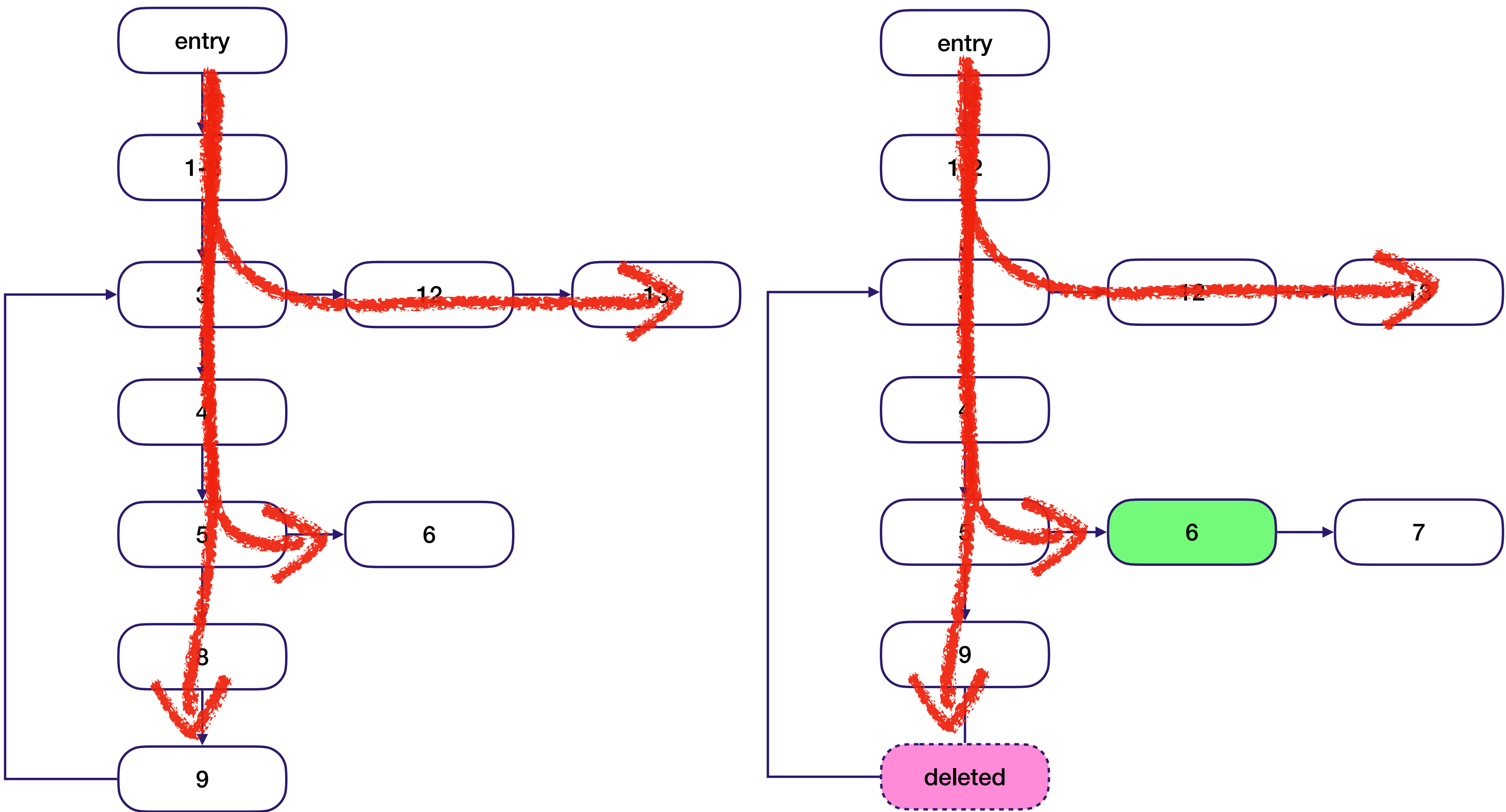
Once difference is found

Add test to selection

selection = { t2 }

Test	Edges Traversed
t1	(entry, 1-2), (1-2, 3), (3, 12), (12, 13)
t2	(entry, 1-2), (1-2, 3), (3, 4), (4, 5), (5, 6)
t3	(entry, 1-2), (1-2, 3), (3, 4), (4, 5), (5, 8), (8, 9), (9, 3), (3, 12), (12, 13)

An Average Example



Start with **selection** = \emptyset

Traverse CFGs in parallel

To spot differences

Once difference is found

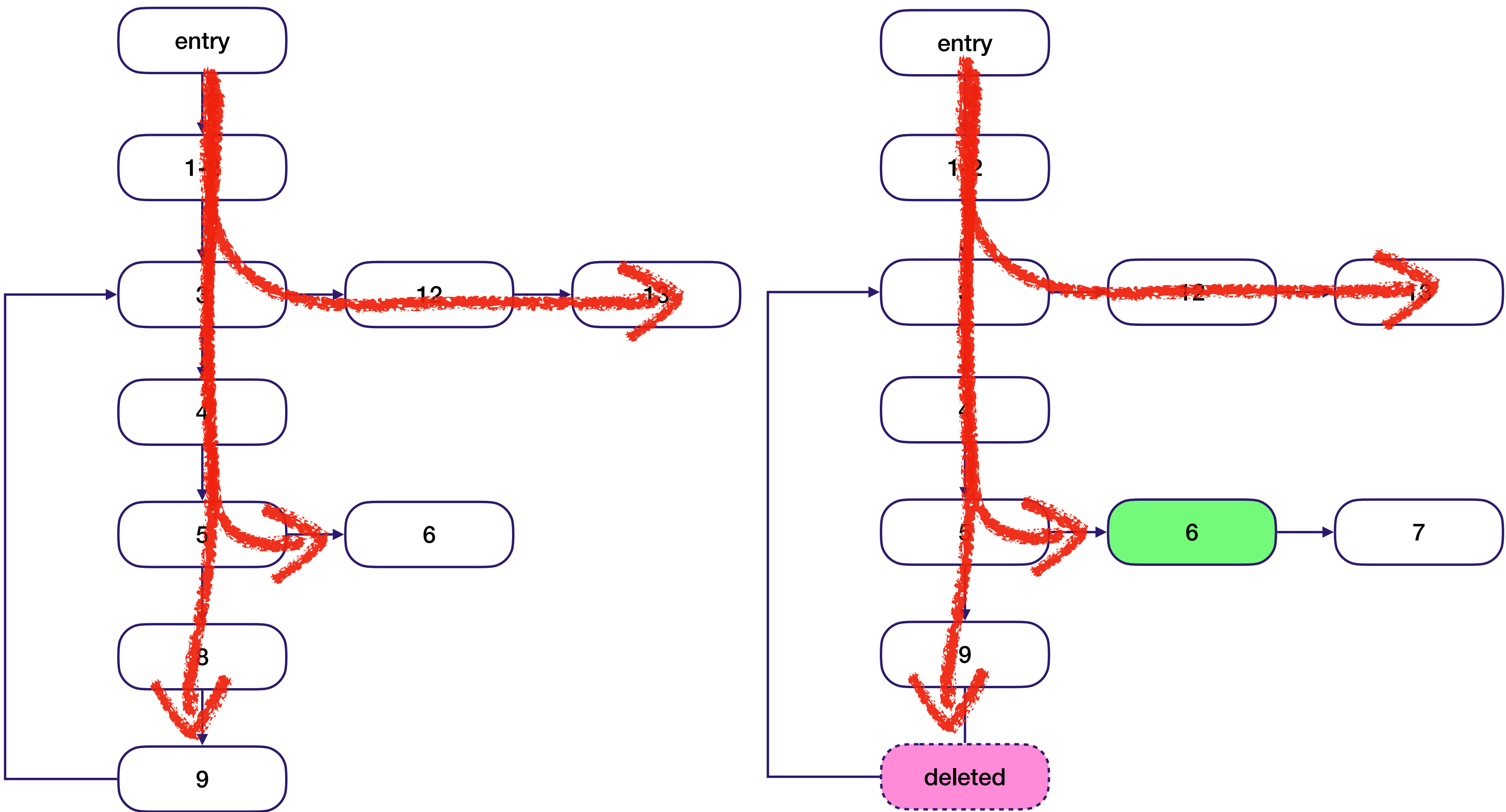
Add test to selection

selection = { t2 }

Repeat recursively

Test	Edges Traversed
t1	(entry, 1-2), (1-2, 3), (3, 12), (12, 13)
t2	(entry, 1-2), (1-2, 3), (3, 4), (4, 5), (5, 6)
t3	(entry, 1-2), (1-2, 3), (3, 4), (4, 5), (5, 8), (8, 9), (9, 3), (3, 12), (12, 13)

An Average Example



Start with **selection** = \emptyset

Traverse CFGs in parallel

To spot differences

Once difference is found

Add test to selection

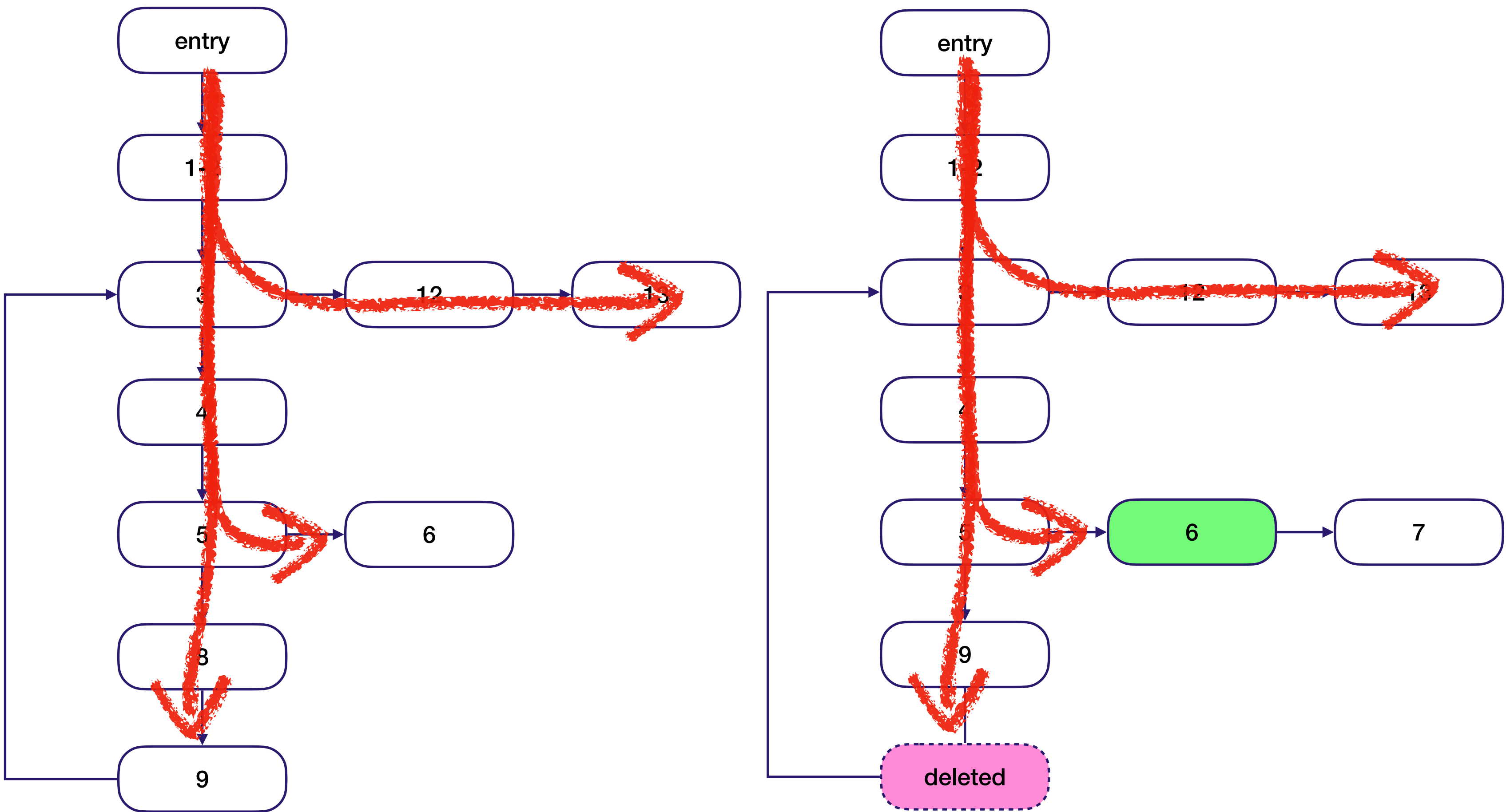
selection = { t2 }

Repeat recursively

selection = { t2, t3 }

Test	Edges Traversed
t1	(entry, 1-2), (1-2, 3), (3, 12), (12, 13)
t2	(entry, 1-2), (1-2, 3), (3, 4), (4, 5), (5, 6)
t3	(entry, 1-2), (1-2, 3), (3, 4), (4, 5), (5, 8), (8, 9), (9, 3), (3, 12), (12, 13)

An Average Example



Test	Edges Traversed
t1	(entry, 1-2), (1-2, 3), (3, 12), (12, 13)
t2	(entry, 1-2), (1-2, 3), (3, 4), (4, 5), (5, 6)
t3	(entry, 1-2), (1-2, 3), (3, 4), (4, 5), (5, 8), (8, 9), (9, 3), (3, 12), (12, 13)

Start with **selection** = \emptyset

Traverse CFGs in parallel

To spot differences

Once difference is found

Add test to selection

selection = { t2 }

Repeat recursively

selection = { t2, t3 }

Complete traversal of CFGs

Research Highlights

Harrold MJ, Gupta R, Soffa ML. **A methodology for controlling the size of a test suite.** ACM Transactions on Software Engineering and Methodology (TOSEM) 2(3):270–285 (1993)

Rothermel G, Untch RH, Chu C, Harrold MJ. **Prioritizing test cases for regression testing.** IEEE Transactions on Software Engineering (TSE) 27(10):929-948 (2001)

Hadi Hemmati: **Chapter Four - Advances in Techniques for Test Prioritization.** Advances in Computing, vol 112: 185-221 (2019).

Rothermel G, Harrold MJ. **A safe, efficient regression test selection technique.** ACM Transactions on Software Engineering and Methodology (TOSEM) 6(2):173–210 (1997)

Summary

Multiple scenarios in which test suites will be used

Continuous Integration helps with automation of test execution

Regression Testing helps find bugs early

Benefits to minimisation, prioritisation and selection

Simple heuristics – Can be expressed as **optimisation problems**

Many different approaches, from simple greedy algorithms to multi-objective evolutionary algorithms