Chapter 7

Graph Coverage

Dr. Brittany Johnson-Matthews
(Dr. B for short)

https://go.gmu.edu/SWE637
Adapted from slides by Jeff Offutt and Bob Kurtz
Graph Coverage

Structures for Modeling Software

- Input Space
- Graphs
- Logic
- Syntax

Applied to

- Source
- Design
- Specs
- Use Cases

Applied to

- Source
- FSMs
- Specs
- DNF

Applied to

- Source
- Models
- Integration
- Input
Covering Graphs

Graphs are the most commonly used structure for testing

Graphs can come from many sources
  - Control flow graphs
  - Design structure
  - Finite state machines and state charts
  - Use cases

*Implementation knowledge* is usually needed (“white box”)
**Definition of a graph**

A set $N$ of *nodes*, where $N \neq \emptyset$

A set $N_0$ of *initial nodes*, where $N_0 \subseteq N$

A set $N_f$ of *final nodes*, where $N_f \subseteq N$

A set $E$ of *edges*, where each edge connects one node to another

Denoted $(n_i, n_j)$ where $i$ is the *predecessor* node and $j$ is the *successor* node
Example Graphs

$N_0 = \{ 1 \}$
$N_f = \{ 4 \}$
$E = \{ (1,2), (1,3), (2,4), (3,4) \}$
Example Graphs

\[ N_0 = \{ 1 \} \]
\[ N_f = \{ 4 \} \]
\[ E = \{ (1,2), (1,3), (2,4), (3,4) \} \]

\[ N_0 = \{ 1, 2, 3 \} \]
\[ N_f = \{ 8, 9, 10 \} \]
\[ E = \{ (1,4), (1,5), (2,5), (3,6), (3,7), (4,8), (5,8), (5,9), (6,2), (6,9), (6,10), (7,10) \} \]
Example Graphs

\[ N_0 = \{1\} \]
\[ N_f = \{4\} \]
\[ E = \{(1,2), (1,3), (2,4), (3,4)\} \]

\[ N_0 = \{1, 2, 3\} \]
\[ N_f = \{8, 9, 10\} \]
\[ E = \{(1,4), (1,5), (2,5), (3,6), (3,7), (4,8), (5,8), (5,9), (6,2), (6,9), (6,10), (7,10)\} \]

\[ N_0 = \{\} \]
\[ N_f = \{4\} \]
\[ E = \{(1,2), (1,3), (2,4), (3,4)\} \]
**Paths in Graph**

Path: a sequence of nodes \([ n_1, n_2, \ldots n_M ]\)

Recall that each pair of nodes is an edge

Length: the number of *edges*

A single node is a path of length 0

Subpath: a subsequence of nodes in \(p\) is a subpath of \(p\)

*A few paths:*

\([1, 4, 8]\)  
\([2, 5, 9, 6]\)  
\([5, 9, 6, 10]\)
Test Paths and SESE Graphs

**Test Path**: a path that starts at an *initial node* and ends at a *final node*

Test paths represent execution of test cases

- Some test paths can be executed by many tests
- Some test paths cannot be executed by any tests

**Single-Entry Single-Exit (SESE) Graphs**: all test paths start at one node and end at one node

$N_0$ and $N_f$ each have exactly one node

Double-diamond graph

Four test paths:
- $[1, 2, 4, 5, 7]$
- $[1, 2, 4, 6, 7]$
- $[1, 3, 4, 5, 7]$
- $[1, 3, 4, 6, 7]$
Visiting and Touring

Visit

a test path $p$ visits node $n$ if $n$ is in $p$,
a test path $p$ visits edge $e$ if $e$ is in $p$

Tour

a test path $p$ tours subpath $q$ if $q$ is a subpath of $p$

Given test path $p = [1, 2, 4, 5, 7]$
$p$ visits nodes 1, 2, 4, 5, 7
$p$ visits edges (1,2), (2,4), (4,5), (5,7)
$p$ tours subpaths [1, 2], [2, 4], [4, 5], [5, 7],
[1, 2, 4], [2, 4, 5], [4, 5, 7], [1, 2, 4, 5], [2, 4, 5, 7],
[1, 2, 4, 5, 7]
**Tests and Test Paths**

**path** \((t)\): the test path executed by test \(t\)

**path** \((T)\): the set of test paths executed by the set of tests \(T\)

Each test executes *exactly one* test path

It is the complete execution from some initial node to some final node
A location (node or edge) in a graph can be reached from another location if there is a sequence of edges from the first location to the second

**Syntactic reach:** a subpath exists in the graph from the first location to the second
    
    This is based only on the graph structure

**Semantic reach:** a test exists that can execute that subpath
    
    This considers the actual implementation logic
Covering Graphs

We use graphs in testing to:

- Develop a model of the software (as a graph)
- Require tests to visit or tour nodes, edges, or subpaths

Test requirements (TRs) describe the properties of test paths

Test criteria are rules that define the test requirements

**DEFINITION**

**Satisfaction** – given a set of test requirements $TR$ for a criterion $C$, a set of tests $T$ satisfies $C$ on a graph if and only if for each test requirement $tr$ in $TR$, there is a test path in $\text{path}(T)$ that meets the test requirement $tr$. 
Structural Coverage Criteria

Structural coverage criteria are defined on a graph only in terms of nodes and edges.

The goal of structural coverage is to ensure that control flow executes successfully.
**Node Coverage**

The first (and simplest) structural coverage criteria requires that each node in a graph be executed.

**Definition**

**Node Coverage (NC)** – test set $T$ satisfies node coverage on graph $G$ if and only if for every syntactically reachable node $n$ in $N$, there is some path $p$ in $\text{path}(T)$ such that $p$ visits $n$.

Or, in terms of test requirements:

**Definition**

**Node Coverage (NC)** – $TR$ contains each reachable node in $G$.

Is node coverage the same as “statement coverage”?
**Edge Coverage**

Edge coverage is slightly stronger than NC

**Definition**

Edge Coverage (EC) – $TR$ contains each reachable path of length up to 1, inclusive, in $G$.

“length up to 1” allows for graphs with one node and no edges.

EC TRs differ from NC TRs only when there is a path with length > 1 and a path with length = 1 between two nodes.

Example: if-then statement

Is edge coverage the same as “branch coverage”? 

[Diagram of a simple if-then statement with nodes 1, 2, and 3, showing the flow of true and false branches.]

16
**Path length “up to 1”?**

A path with only one node has no edges

It may seem trivial, but formally edge coverage must require node coverage on this graph, otherwise EC will not subsume NC

We will see the same issue later for edge-pair coverage when a graph has only one edge
Covering Multiple Edges

Edge-pair coverage requires pairs of edges, or subpaths of length=2

**Edge-Pair Coverage (EPC)** – TR contains each reachable path of length up to 2, inclusive, in G.

“length up to 2” allows for graphs with two nodes and one edge

Edge-Pair Coverage:
TR = { [1,3,4], [1,3,5], [2,3,4], [2,3,5] }
Covering Multiple Edges

This suggests an obvious extension to...

**Complete Path Coverage (CPC)** – $TR$ contains all paths in $G$.

But this is impossible if the graph has a loop

A weak compromise is to let the tester decide which paths to test

**Specified Path Coverage (SPC)** – $TR$ contains a set $S$ of test paths, where $S$ is supplied as a parameter.
**Structural Coverage Example**

**Node Coverage:**
TR = \{ 1, 2, 3, 4, 5 \}
Test paths = [ 1, 2, 3, 4, 3, 5 ]

**Edge Coverage:**
TR = \{ (1,2), (1,3), (2,3), (3,4), (4,3), (3,5) \}
Test paths = [ 1, 2, 3, 4, 3, 5 ], [ 1, 3, 5 ]

**Edge-Pair Coverage:**
TR = \{ [1,2,3], [1,3,4], [1,3,5], [2,3,4], [2,3,5], [3,4,3], [4,3,4], [4,3,5] \}
Test paths = [ 1, 2, 3, 4, 3, 5 ], [ 1, 3, 4, 3, 4, 3, 5 ], [ 1, 3, 5 ], [ 1, 2, 3, 5 ]
Complete Path Coverage:
Test paths = [ 1, 3, 5 ], [1, 2, 3, 5 ],
[ 1, 2, 3, 4, 3, 5 ],
[ 1, 2, 3, 4, 3, 4, 3, 5 ],
[ 1, 2, 3, 4, 3, 4, 3, 4, 3, 5 ],
[ 1, 2, 3, 4, 3, 4, 3, 4, 3, 4, 3, 5 ],
…
Handling Loops in Graphs

If a graph contains a loop, then it has an infinite number of paths.

Complete path coverage is infeasible.

SPC is not satisfactory because the results are subjective and vary with the tester.

Attempts to deal with loops:

- 1980s: execute loops exactly once
- 1990s: execute loops 0, 1, >1 times
- 2000s: prime paths (touring, sidetrips, detours)
Simple Paths and Prime Paths

**Simple path**: a path from node $n_i$ to $n_j$ is *simple* if no node appears more than once, except that the first and last nodes may be the same.

A simple path has no loops within it, but a loop is itself a simple path.

**Prime path**: a simple path that does not appear as a proper subpath of any other simple path.

Simple Paths:
- $[1,2,4]$, $[1,3,4]$, $[2,4,1]$, $[3,4,1]$, $[4,1,2]$, $[4,1,3]$
- $[1,2,4,1]$, $[1,3,4,1]$, $[2,4,1,2]$, $[2,4,1,3]$
- $[3,4,1,2]$, $[3,4,1,3]$, $[4,1,2,4]$, $[4,1,3,4]$

Prime Paths:
- $[1,2,4,1]$, $[1,3,4,1]$, $[2,4,1,2]$, $[2,4,1,3]$
- $[3,4,1,2]$, $[3,4,1,3]$, $[4,1,2,4]$, $[4,1,3,4]$
Prime Path Coverage

A simple, elegant and finite criterion that requires loops to be executed as well as skipped

**DEFINITION**

Prime Path Coverage (PPC) – TR contains each prime path in G.

Will tour all paths of length 0, 1, …, N

Subsumes node and edge coverage
**PPC Does Not Subsume EPC**

If a node $j$ has an edge to itself (a *self edge*), then edge-pair coverage requires $[i, j, j]$ and $[j, j, k]$.

Neither $[i, j, j]$ nor $[j, j, k]$ are *simple paths* and thus not *prime paths*.

**EPC Requirements:**

$$TR = \{ [1,2,3], [1,2,2], [2,2,2], [2,2,3] \}$$

**PPC Requirements:**

$$TR = \{ [1,2,3], [2,2] \}$$
Prime Path Example

Simple Paths (16):
- [1,2], [1,3], [2,3], [3,4], [3,5], [4,3], [1,2,3], [1,3,4], [1,3,5], [2,3,4], [2,3,5], [3,4,3], [4,3,4], [4,3,5], [1,2,3,4], [1,2,3,5]

Prime Paths (7):
- [1,3,4], [1,3,5], [3,4,3], [4,3,4], [4,3,5], [1,2,3,4], [1,2,3,5]

Loop 0 times
Loop at least once
Loop more than once
Touring

Tour: a test path $p$ tours subpath $q$ if $q$ is a subpath of $p$

Touring the prime path [1,2,3,5,6] without sidetrips or detours
Touring with Sidetrips

Tour with sidetrips: a test path $p$ tours subpath $q$ with sidetrips if and only if every edge in $q$ is also in $p$ in the same order.

The tour can sidetrip from node $n_i$ as long as it comes back to $n_i$.
Touring with Detours

Tour with detours: a test path $p$ tours subpath $q$ with detours if and only if every node in $q$ is also in $p$ in the same order.

A tour can detour from node $n_i$ as long as it comes back to the prime path at a successor of $n_i$.

Touring the prime path $[1,2,3,5,6]$ with a detour to node 4.
**Infeasible Test Requirements**

An *infeasible* test requirement cannot be satisfied

- Unreachable statement (dead code)
- Subpath that can only be executed with a contradiction \((x > 0 \text{ and } x < 0)\)

Most test criteria have some infeasible test requirements

It is usually *undecidable* whether all test requirements are feasible

When sidetrips are not allowed, structural criteria typically have *more* infeasible requirements

However, allowing sidetrips weakens the test criteria
Best effort touring is a practical compromise

Satisfy as many test requirements as possible without sidetrips

Allow sidetrips to try to satisfy remaining test requirements
Data Flow Coverage

Data flow coverage criteria also require the graph to be annotated with references to variables.

The goal of data flow coverage is to ensure that values are computed and used correctly.

Definition (Def): a location where a variable’s value is set

Use: a location where a variable’s value is accessed
The values set in each *def* should *reach* at least one, some, or all possible *uses*.

**Defs:**
- def(1) = \{ x \}
- def(5) = \{ z \}
- def(6) = \{ z \}

**Uses:**
- use(5) = \{ x \}
- use(6) = \{ x \}
DU Pairs

def(n) or def(e): the set of variables that are defined by node n or edge e

use(n) or use(e): the set of variables that are used by node n or edge e

DU pair: a pair of locations \((l_i, l_j)\) such that a variable \(v\) is defined at \(l_i\) and used at \(l_j\)
**DU Paths**

**Def-clear**: a path from $l_i$ to $l_j$ is *def-clear* with respect to variable $v$ if $v$ is not given another value on any of the nodes or edges of the path.

**Reach**: if there is a def-clear path from $l_i$ to $l_j$ with respect to $v$, the def of $v$ at $l_i$ *reaches* the use of $v$ at $l_j$.

**DU-path**: a simple subpath that is def-clear with respect to $v$ from a def of $v$ to a use of $v$. 
Touring DU-Paths

A test path $p$ DU-tours subpath $d$ with respect to $v$ if $p$ tours $d$ and the subpath taken is def-clear with respect to $v$

Sidetrips can be used as with previous touring

Three obvious criteria

Use every def
Get to every use
Follow all DU-paths
Data flow test criteria

First, ensure every def reaches a use

**Definition**

**All-Defs Coverage (ADC)** – for each set of DU-paths $S = du(n_v)$, $TR$ contains at least one path $d$ in $S$.

Then, ensure every def reaches all uses

**Definition**

**All-Uses Coverage (AUC)** – for each set of DU-paths to uses $S = du(n_i, n_j, v)$, $TR$ contains at least one path $d$ in $S$.

Finally, cover all the paths between defs and uses

**Definition**

**All-DU-Paths Coverage (ADUPC)** – for each set $S = du(n_i, n_j, v)$, $TR$ contains every path $d$ in $S$. 
**Data Flow Testing Example**

All-defs for $x$:
- $[1, 2|3, 4, 5|6]$ 

All-uses for $x$:
- $[1, 2|3, 4, 5]$, $[1, 2|3, 4, 6]$ 

All-DU-paths for $x$:
- $[1, 2, 4, 5]$, $[1, 3, 4, 5]$, $[1, 2, 4, 6]$, $[1, 3, 4, 6]$
Graph Coverage Criteria Subsumption

A test criterion $C_1$ subsumes $C_2$ if and only if every set of test cases that satisfies criterion $C_1$ also satisfies $C_2$.

**DEFINITION**

- Complete Path Coverage (CPC)
- Prime Path Coverage (PPC)
- Edge-Pair Coverage (EPC)
- Complete Round-Trip Coverage (CRTC)
- Simple Round-Trip Coverage (SRTC)
- All DU-Paths Coverage (ADUPC)
- All-Uses Coverage (AUC)
- All-Defs Coverage (ADC)
- Edge Coverage (EC)
- Node Coverage (NC)

Subsumes all others.
Summary

Graphs are a powerful abstraction for designing tests

Various criteria allow cost/benefit trades

Graphs appear in many situations in software

We’ll explore this further next week