Graph: Structural Coverage Criteria
(Intro, NC, EC, EPC)

CS 3250
Software Testing

[Ammann and Offutt, “Introduction to Software Testing,” Ch. 7]
Structures for Criteria-Based Testing

Four structures for modeling software

Input space

- Graph
  - Source
  - Design
  - Specs
  - Use cases
  - Applied to: R--R

Logic

- Source
- Specs
- FSMs
- DNF
- Applied to: RI-R

Syntax

- Source
- Models
- Integration
- Inputs
- Applied to: RIPR
Today’s Objectives

- Investigate some of the most widely known test coverage criteria
- Understand basic theory of graph
  - Generic view of graph without regard to the graph’s source
- Understand how to use graph to define criteria and design tests
  - Node coverage (NC)
  - Edge coverage (EC)
  - Edge-pair coverage (EPC)
- Graph derived from various software artifacts (coming soon)
Overview

- Graphs are the most commonly used structure for testing
- Graphs can come from many sources
  - Control flow graphs from source
  - Design structures
  - Finite state machine (FSM)
  - Statecharts
  - Use cases
- The graph is not the same as the artifact under test, and usually omits certain details
- Tests must **cover** the graph in some way
  - Usually traversing specific portions of the graph
Graph: Nodes and Edges

• **Node** represents
  • Statement
  • State
  • Method
  • Basic block

• **Edge** represents
  • Branch
  • Transition
  • Method call
Basic Notion of a Graph

• Nodes:
  • \( N \) = a set of nodes, \( N \) must not be empty

• Initial nodes
  • \( N_0 \) = a set of initial nodes, must not be empty
  • Single entry vs. multiple entry

• Final nodes
  • \( N_f \) = a set of final nodes, must not be empty
  • Single exit vs. multiple exit

• Edges:
  • \( E \) = a set of edges, each edge from one node to another
  • An edge is written as \((n_i, n_j)\)
    • \( n_i \) is predecessor, \( n_j \) is successor
  
Every test must **start** in some initial node, and **end** in some final node
Note on Graphs

- The concept of a final node depends on the kind of software artifact the graph represents.
- Some test criteria require tests to end in a particular final node.
- Some test criteria are satisfied with any node for a final node (i.e., the set $N_f = \text{the set } N$).
Example Graph

- **Node**
  \[ N = \{1, 2, 3, 4\} \]
  \[ N_0 = \{1\} \]
  \[ N_f = \{4\} \]

- **Edge**
  \[ E = \{(1,2), (1,3), (2,4), (3,4)\} \]

Is this a graph?

Single-Entry, Single-Exit (SESE)

\[ N = \{1\} \]
\[ N_0 = \{1\} \]
\[ N_f = \{1\} \]
\[ E = \{\} \]
Example Graph

- **Node**
  \[ N = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\} \]
  \[ N_0 = \{1, 2, 3\} \]
  \[ N_f = \{8, 9, 10\} \]

- **Edge**
  \[ E = \{(1,4), (1,5), (2,5), (6,2), (3,6), (3,7), (4,8), (5,8), (5,9), (6,10), (7,10), (9,6)\} \]
Example Graph

- **Node**
  
  \[N = \{1, 2, 3, 4\}\]

  \[N_0 = \{\}\]

  \[N_f = \{4\}\]

- **Edge**

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

Not valid graph – no initial nodes
Not useful for generating test cases
Paths in Graphs

- **Path $p$**
  - A sequence of nodes, $[n_1, n_2, ..., n_M]$
  - Each pair of adjacent nodes, $(n_i, n_{i+1})$, is an edge

- **Length**
  - The number of edges
  - A single node is a path of length 0

- **Subpath**
  - A subsequence of nodes in $p$ (possibly $p$ itself)
Example Paths

- **Paths**
  - \([1, 4, 8]\)
  - \([2, 5, 8]\)
  - \([2, 5, 9]\)
  - \([2, 5, 9, 6, 10]\)
  - \([3, 6, 10]\)
  - \([3, 7, 10]\)
  - \([3, 6, 2, 5, 9]\)
  - \([2, 5, 9, 6, 2]\)

- **Cycle** – a path that begins and ends at the same node
Example Paths

- Invalid paths
  - [1, 8]
  - [4, 5]
  - [3, 7, 9]

Invalid path – a path where the two nodes are not connected by an edge.
def template(num1, num2):
    result = ""
    if num1 == 0:
        result = "num1 is 0"
    elif num1 == 1:
        result = "num1 is 1"
        if num2 > 3:
            result = " num2 > 3"
    elif num2 > 4:
        result = "This will never run"
    else:
        result = " num2 <= 3"
    else:
        result = "num1 is not 0 or 1"
    return result
Invalid Paths

- Many test criteria require inputs that start at one node and end at another. – This is only possible if those nodes are connected by a path.

- When applying these criteria on specific graphs, we sometimes find that we have asked for a path that for some reason cannot be executed.

- Example: a path may demand that a loop be executed zero time, where the program always executed the loop at least once.

- This problem is based on the semantics of the software artifact that the graph represents.

- For now, let’s emphasize only the syntax of the graph
Graph and Reachability

• A location in a graph (node or edge) can be reached from another location if there is a sequence of edges from the first location to the second

• Syntactically reachable
  • There exists a subpath from node $n_i$ to $n$ (or to edge $e$)

• Semantically reachable
  • There exists a test that can execute that subpath
Example: Reachability

- From node 1
  - Possible to reach all nodes except nodes 3 and 7

- From node 5
  - Possible to reach all nodes except nodes 1, 3, 4, and 7

- From edge (7, 10)
  - Possible to reach nodes 7 and 10 and edge (7, 10)

Some graphs (such as finite state machines) have explicit edges from a node to itself, that is \((n_i, n_i)\)
Test Paths

- A path that starts at an initial node and end at a final node

- A test path represents the execution test cases
  - Some test paths can be executed by many test cases
  - Some test paths cannot be executed by any test cases
  - Some test paths cannot be executed because they are infeasible
SESE Graphs

- **SESE** (Single-Entry-Single-Exit) graphs
  - The set $N_0$ has exactly one node ($n_0$)
  - The set $N_f$ has exactly one node ($n_f$), $n_f$ may be the same as $n_0$
  - $n_f$ must be syntactically reachable from every node in $N$
  - No node in $N$ (except $n_f$) be syntactically reachable from $n_f$
    (unless $n_0$ and $n_f$ are the same node)

Double-diamonded graph
(two if-then-else statements)

4 test paths
- $[1, 2, 4, 5, 7]$
- $[1, 2, 4, 6, 7]$
- $[1, 3, 4, 5, 7]$
- $[1, 3, 4, 6, 7]$
Visiting

- A test path \( p \) visits node \( n \) if \( n \) is in \( p \)
- A test path \( p \) visits edge \( e \) if \( e \) is in \( p \)

Consider path \([1, 2, 4, 5, 7]\)
Visits node: 1, 2, 5, 4, 7
Visits edge: (1,2), (2,4), (4,5), (5,7)
Touring

- A test path \( p \) tours subpath \( q \) if \( q \) is a subpath of \( p \)

Node \( N = \{1, 2, 3, 4, 5, 6, 7\} \)

Edge \( E = \{(1,2), (1,3), (2,4), (3,4), (4,5), (4,6), (5,7), (6,7)\} \)

(Each edge is technically a subpath)

Consider a test path \([1, 2, 4, 5, 7]\)

Visit notes: \(1, 2, 4, 5, 7\)

Visit edges: \((1,2), (2,4), (4,5), (5,7)\)

Tours subpaths: \([1,2,4,5,7], [1,2,4,5], [2,4,5,7], [1,2,4], [2,4,5], [4,5,7], [1,2], [2,4], [4,5], [5,7]\)

Any given path \( p \) always tours itself
Mapping: Test Cases – Test Paths

- \( \text{path}(t) = \) Test path executed by test case \( t \)
- \( \text{path}(T) = \) Set of test paths executed by set of tests \( T \)
- Test path is a complete execution from a start node to a final node

- **Minimal** set of test paths = the fewest test paths that will satisfy test requirements
  - Taking any test path out will no longer satisfy the criterion
Mapping: Test Cases – Test Paths

**Deterministic software:** test always executes the same test path

- test 1
- test 2
- test 3
  - many-to-one
  - Test Path 1

**Non-deterministic software:** the same test can execute different test paths

- test 1
- test 2
- test 3
  - many-to-many
  - Test Path 1
  - Test Path 2
  - Test Path 3
Example Mapping
Test Cases – Test Paths

Test case t1: (a=0, b=1) \rightarrow [ Test path p1: 1, 2, 4, 3]
Test case t2: (a=1, b=1) \rightarrow [ Test path p2: 1, 4, 3]
Test case t3: (a=2, b=1) \rightarrow [ Test path p3: 1, 3]

[AO, page 111, Figure 7.5]
Graph Coverage Criteria

Graph coverage criteria define test requirements TR in terms of properties of test paths in a graph G

Steps:

1. Develop a model of the software as a graph
2. A test requirement is met by visiting a particular node or edge or by touring a particular path

Test requirements (TR)

- Describe properties of test paths

Test criterion

- Rules that define test requirements
Graph Coverage Criteria

Satisfaction

• Given a set $TR$ of test requirements for a criterion $C$, a set of tests $T$ satisfies $C$ on a graph if and only if for every test requirement in $TR$, there is a test path in $\text{path}(T)$ that meets the test requirement $tr$

Two types

1. **Structural coverage criteria**
   - Define a graph just in terms of nodes and edges

2. **Data flow coverage criteria**
   - Requires a graph to be annotated with references to variables
Graph Coverage Criteria

Structural Coverage Criteria

- Node Coverage (NC)
  - Statement coverage
- Edge Coverage (EC)
  - Branch coverage
- Edge-Pair Coverage (EPC)
- Complete Path Coverage (CPC)
- Prime Path Coverage (PPC)

Data Flow Coverage Criteria

- All-Defs Coverage (ADC)
- All-Uses Coverage (AUC)
- All-du-Paths Coverage (ADUPC)
Node Coverage (NC)

Node Coverage (NC) means that every reachable node in the graph is contained in the test set. Formally, if $TR$ is the set of nodes covered by a test set $T$, then $TR$ contains each reachable node in the graph $G$.

Given a graph $G$ with nodes $N = \{1, 2, 3, 4, 5, 6, 7\}$ and edges $E = \{(1,2), (1,3), (2,4), (3,4), (4,5), (4,6), (5,7), 6,7\}$, the test set $TR = \{1, 2, 3, 4, 5, 6, 7\}$ satisfies Node Coverage.

Test paths $p_1 = [1, 2, 4, 5, 7]$ and $p_2 = [1, 3, 4, 6, 7]$ cover all nodes in the graph.

If a test set $T = \{t1, t2\}$, where $\text{path}(t1) = p1$ and $\text{path}(t2) = p2$, then $T$ satisfies Node Coverage on $G$.
**Edge Coverage (EC)**

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

Node $N = \{1, 2, 3, 4, 5, 6, 7\}$

Edge $E = \{(1,2), (1,3), (2,4), (3,4), (4,5), (4,6), (5,7), (6,7)\}$

Test path $p_1 = [1, 2, 4, 5, 7]$  
Test path $p_2 = [1, 3, 4, 6, 7]$

If a test set $T = \{t_1, t_2\}$,  
where path($t_1$) = $p_1$ and path($t_2$) = $p_2$,  
Then $T$ satisfies Edge Coverage on G
Difference between NC and EC

Node $N = \{1, 2, 3\}$
Edge $E = \{(1, 2), (1, 3), (2, 3)\}$

NC: $TR = \{1, 2, 3\}$
Test path = [1, 2, 3]

EC: $TR = \{(1, 2), (1, 3), (2, 3)\}$
Test paths = [1, 2, 3], [1, 3]

NC and EC are only different when there is an edge and another subpath between a pair of nodes (as in an “if-else” statement).
Edge-Pair Coverage (EPC)

EPC: TR contains each reachable path of length up to 2, inclusive, in G

“length up to 2” – allows for graphs that have 0, 1, or 2 edges

Node $N = \{1, 2, 3, 4, 5, 6, 7\}$

Edge $E = \{(1,2), (1,3), (2,4), (3,4), (4,5), (4,6), (5,7), (6,7)\}$

$TR = \{(1,2,4), (1,3,4), (2,4,5), (2,4,6), (3,4,5), (3,4,6), (4,5,7), (4,6,7)\}$

Test path $p1 = [1, 2, 4, 5, 7]$  
Test path $p2 = [1, 3, 4, 5, 7]$  
Test path $p3 = [1, 2, 4, 6, 7]$  
Test path $p4 = [1, 3, 4, 6, 7]$

EPC requires pairs of edges, or subpaths of length 2 – covering multiple edges
Graph Coverage Criteria Subsumption

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