Phase 3 – Control Flow Graph (CFG) Structural Similarity Analysis

1. Overall Architecture

Phase 3 introduces Control Flow Graph (CFG) analysis to enhance structural and behavioral similarity detection. The implementation is organized into three primary layers: CFG data structures, CFG construction from AST, and CFG comparison using multiple structural and behavioral metrics.

Layer 1 – CFG Structure: Includes CFGNode and CFG classes with successor and predecessor relationships.

Layer 2 – CFG Construction: CFGBuilder converts AST representations into CFG fragments.

Layer 3 – CFG Comparison: CFGComparator evaluates similarity using five weighted metrics.

2. CFG Data Structures

Each CFGNode contains an identifier, a label, and lists of successor and predecessor nodes. The graph is directed, and edges are represented implicitly through successor relationships.

The CFG class stores the entry node, exit node, and a complete list of nodes. This enables traversal, edge counting, cycle detection, and execution path extraction.

3. CFG Construction (CFGBuilder)

The CFGBuilder transforms AST constructs into structured control-flow fragments. Each fragment consists of an entry node and exit node, following classical compiler construction principles.

3.1 Simple Statements

Assignments, returns, and function calls are modeled as single nodes where entry equals exit.

3.2 Sequential Blocks

For blocks and program sequences, fragments are connected sequentially by linking the exit of the previous statement to the entry of the next, forming linear execution flow.

3.3 IF Statements

IF constructs create a condition node branching into then and optional else fragments, which reconverge at a merge node. This models structured conditional control flow.

3.4 WHILE Loops

WHILE loops generate a condition node connected to the body fragment. The body reconnects to the condition, forming a cycle, and the condition connects to an exit node.

3.5 Simplifications

Condition expressions are abstracted and not represented as separate nodes. Edges are not labeled true/false, and advanced control mechanisms like break/continue are not modeled.

4. CFG Similarity Metrics (CFGComparator)

The comparator combines five metrics into a weighted similarity score.

4.1 Node Count Similarity (Weight: 0.15)

Computed as 1 - |n1 - n2| / max(n1, n2). Measures structural size similarity.

4.2 Edge Count Similarity (Weight: 0.15)

Compares the number of control-flow transitions between programs.

4.3 Label Frequency Similarity (Weight: 0.20)

Builds frequency maps of node labels and computes normalized difference. Captures structural composition such as number of IF, WHILE, RETURN nodes.

4.4 Cycle Detection (Weight: 0.10)

Uses DFS to determine presence of cycles (loops). Returns 1.0 if both graphs either contain or do not contain cycles.

4.5 Execution Path Similarity (Weight: 0.40)

Extracts execution paths from entry to exit and computes Jaccard similarity between the two path sets. Depth is limited to prevent infinite traversal.

5. Weighting Justification

Execution paths are given highest weight because they capture behavioral similarity. Label frequency captures structural composition, while node and edge counts capture size characteristics. Cycle detection provides binary loop presence detection.

6. Strengths of Phase 3

The system models structured control flow, detects loops, incorporates behavioral analysis, and uses graph traversal algorithms. It significantly improves over pure AST comparison.

7. Limitations

Limitations include absence of true/false edge labeling, limited path depth, no graph isomorphism detection, and lack of semantic analysis.

8. Complexity Analysis

CFG Construction: O(N) where N is number of AST nodes.

Cycle Detection: O(V + E).

Path Extraction: Potentially exponential but depth-limited.

Overall comparison complexity remains practical for small to medium programs.

Example:

input1.txt

```
1   int main() {
2       int i = 0;
3       for (i = 0; i < 5; i = i + 1) {
4
5       }
6       return i;
7   }
8
```

input2.txt

```
1   int main() {
2       int i = 0;
3       while (i < 5) {
4           i = i + 1;
5       }
6       return i;
7   }
8
```

```
PROGRAM
  FUNCTION
    VAR_DECL
      CONSTANT
    FOR
      ASSIGN
        IDENTIFIER
        CONSTANT
      BIN_OP
        IDENTIFIER
        CONSTANT
      ASSIGN
        IDENTIFIER
        BIN_OP
          IDENTIFIER
          CONSTANT
    RETURN
      IDENTIFIER
PROGRAM
  FUNCTION
    VAR_DECL
      CONSTANT
    WHILE
      BIN_OP
        IDENTIFIER
        CONSTANT
      ASSIGN
        IDENTIFIER
        BIN_OP
          IDENTIFIER
          CONSTANT
    RETURN
      IDENTIFIER
```

```
      IDENTIFIER

Max Subtree Similarity: 100.00%
Weighted AST Similarity: 71.79%
AST Decision: Similar
CFG Similarity: 0.5416666666666667
CFG 1:
Node 0 (VAR_DECL) -> 1
Node 1 (ASSIGN) -> 2
Node 2 (LOOP_COND) -> 3 5
Node 3 (EMPTY) -> 4
Node 4 (ASSIGN) -> 2
Node 5 (LOOP_EXIT) -> 6
Node 6 (RETURN) ->

CFG 2:
Node 7 (VAR_DECL) -> 8
Node 8 (LOOP_COND) -> 9 10
Node 9 (ASSIGN) -> 8
Node 10 (LOOP_EXIT) -> 11
Node 11 (RETURN) ->
```