Algorithm 1 Depth-First-Search

```
procedure DEPTH-FIRST-SEARCH(model)
    $nodes \leftarrow model.getNodes()$
    $adjacencyMatrix \leftarrow \emptyset$
    $visitedNodes \leftarrow \emptyset$
    $minimizedModel \leftarrow \emptyset$
    for $i \leftarrow 0, nodes.Size()$ do
        $node \leftarrow nodes[i]$
        $depthFirstSearch(node, visitedNodes)$
    end for
end procedure
```

Algorithm 2 Depth-First-Search-Util

```
procedure DEPTH-FIRST-SEARCH-UTIL(vertex, visitedNodes)
    if $(!visitedNodes[vertex])$ then
        $visitedNodes[vertex] \leftarrow true$
        $neighbors \leftarrow vertex.getOutgoingEdges()$
        $createMetrics(vertex, neighbors)$
        for $i \leftarrow 0, neighbors.Size()$ do
            $neighbor \leftarrow neighbors[i]$
            $depthFirstSearch(neighbor, visitedNodes)$
        end for
    end if
end procedure
```

Algorithm 3 Create-Metrics

```
1: procedure CREATE-METRICS(vertex, neighbors)
2:     $createStructureMetric(vertex, neighbors)$
3:     $createEquationMetric(vertex, neighbors)$
4:     $createActivityMetric(vertex, neighbors)$
5:     $createTimeConstraintMetric(vertex, neighbors)$
6:     $createPropagationMetric(vertex, neighbors)$
7:     $createGraphEditDistance(vertex, neighbors)$
8: end procedure
```

Algorithm 4 Create-Structure-Metric

```
1: procedure CREATE-STRUCTURE-METRIC(vertex, neighbors)
2:     $vertexLabel \leftarrow vertex.getLabel()$
3:     $neighborLabels \leftarrow neighbors.getLabel()$
4:     $createAdjacencyMatrix(vertexLabel, neighborLabels)$
5: end procedure
```

Algorithm 5 Create-Equation-Metric

```
1: procedure CREATE-EQUATION-METRIC(vertex, neighbors)
2:     $vertexEquation \leftarrow vertex.getEquation()$
3:     $neighborEquations \leftarrow neighbors.getEquations()$
4:     $createEquationMatrix(vertexEquation, neighborEquations)$
5: end procedure
```

Algorithm 6 Create-Activity-Metric

```
1: procedure CREATE-ACTIVITY-METRIC(vertex, neighbors)
2:     $vertexActiveTime \leftarrow vertex.getEquation.getDuration()$
3:     $neighborActiveTimes \leftarrow neighbors.getEquations().getDurations()$
4:     $createActivityMatrix(vertexActiveTime, neighborActiveTimes)$
5: end procedure
```

Algorithm 7 Create-Time-Constraint-Metric

```
1: procedure CREATE-TIME-CONSTRAINT-METRIC(vertex, neighbors)
2:     $vertexConstraint \leftarrow vertex.getEquation.getGuards()$
3:     $neighborConstraints \leftarrow neighbors.getEquations().getGuards()$
4:     $createTimeConstraintMatrix(vertexConstraint, neighborConstraints)$
5: end procedure
```

Algorithm 8 Trace-Analyzer

```
1: procedure ANALYZE-TRACE(traceArray)
2:     for $i \leftarrow 0, traceArray.Size()$ do
3:         $trace \leftarrow traceArray[i]$
4:         for $j \leftarrow 0, trace.Size()$ do
5:             $traceData \leftarrow trace[j]$
6:             if $traceData.isNumeric()$ then
7:                 buffer ← bufferInit(timeStep, bufferSize, traceData)
8:                 equationTrace ← fitData(timeStep, buffer, traceData)
9:                 $incrementalConstruction$(equationTrace);
10:            end if
11:        end for
12:    end for
13: end procedure
```

Algorithm 9 Procedure Incremental-Construction. Variable recreatedModel is initialized as empty. Prefix OM resembles the original model, while RM the recreated model.

```
1: procedure INCREMENTAL-CONSTRUCTION(equationTrace)
2:     for $i \leftarrow 0, equationTrace.Size()$ do
3:         $equation \leftarrow equationTrace[i]$
4:         $recreatedModel.addEquation(equation)$
5:         $calculateStructureMetric(equation, structure\_OM, structure\_RM)$
6:         $calculateEquationMetric(equation, functionality\_OM, functionality\_RM)$
7:         $calculateActivityMetric(equation, activity\_OM, activity\_RM)$
8:         $calculateTimeConstraintMetric(equation, timeConstraint\_OM, timeConstraint\_RM)$
9:     end for
10:    $calculateGraphEditDistance(traceMetricsArray)$
11:    $calculatePropagationMetric()$
12: end procedure
```

Algorithm 10 Calculate-Structure-Metric

```
1: procedure CALCULATE-STRUCTURE-METRIC(equation)
2:     $equationLabel \leftarrow equation.getLabel()$
3:     $findOriginalStructure(equationLabel)$
4:     $calculateIncrementalScore(equationLabel)$
5:     $updateRecreatedStructure(equationLabel)$
6: end procedure
```