University of Coimbra

System Modelling and Analysis

Path-set selection in control-flow testing

António Lima
2011166926

Inês Petronilho
2012137900

Pedro Janeiro
2012143629

November 16, 2015

Contents

1 Objective

In this assignment, we were asked to produce integer linear programming (***ILP***) models to enumerate paths in a control flow graph and to find exact solutions to the ***optimal weighted path-set selection problem***.

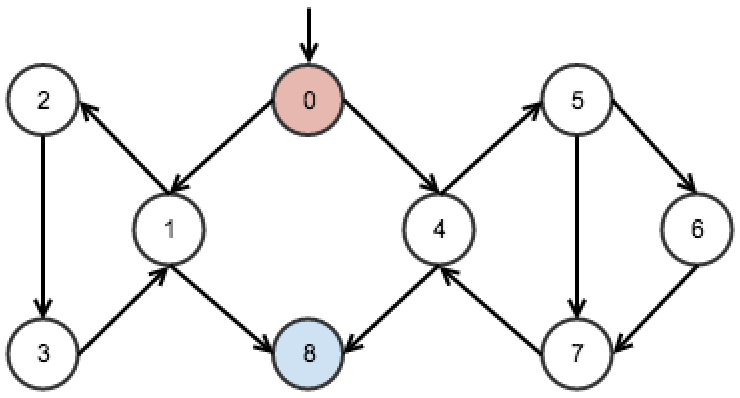
2 Data

Assuming that the data is passed as the set of edges that make up the control flow graph, and that the source and target nodes are denoted by the smallest and largest node number, respectively.

For example, ***cfg0_1.dat*** includes the following edges

[0 1], [0 4], [1 2], [1 8], [2 3], [3 1], [4 5], [4 8], [5 6], [5 7], [6 7], [7 4]

which represent the following graph



Since this is a rather short and test-case complete (it has a simple cycle and a larger cycle which can be broken down into a smaller cycle) graph we will use it for the following examples. All of the provided examples (cf1, cfg2, cfg3, cfg4) are either acyclic or far too complex to analyse quickly.

3 Test-path generation

3.1 Shortest Paths

As suggested, in order to determine the non-basic edges and, consequently, the basis edges we needed to determine the shortest path from each node to the target node. Using ***spp.mod***, an example that came with ***GLPK***, the following formulation guaranteed an ***acyclic first shortest path*** by specifying that each node must have one incoming edge and one outgoing edge, giving special care to the source and target nodes:

$$min \sum_{(i,j)\in E} c[i,j] * x[i,j]$$

$$s.t.: \sum_{(j,i)\in E} x[j,i] + t1 = \sum_{(i,j)\in E} x[i,j] + t2, \forall i = 0 \ldots n$$

$$t_1 = \begin{cases} 1, & i = s \\ 0, & \text{otherwise} \end{cases}$$

$$t_2 = \begin{cases} 1, & i = t \\ 0, & \text{otherwise} \end{cases}$$

For this assignment all edges had unitary cost but, as can be seen in the ***ILP*** model above, this could be applied to any cost (negative, zero, positive).

Keeping ***cfg0_1.dat*** in mind, our model yields the the following shortest path edges for each source node:

- ***0***: [0 4], [4 8]
- ***1***: [1 8]
- ***2***: [2 3], [3 1], [1 8]
- ***3***: [3 1], [1 8]
- ***4***: [4 8]
- ***5***: [5 7], [7 4], [4 8]
- ***6***: [6 7], [7 4], [4 8]
- ***7***: [7 4], [4 8]

These in turn can be post-processed in order to obtain the set of

- ***non-basic edges***: [0, 4], [1, 8], [2, 3], [3, 1], [4, 8], [5, 7], [6, 7], [7, 4]
- ***basis edges***: [0, 1], [1, 2], [4, 5], [5, 6]

The basis edges are a counter measure which will be used later on

3.2 Cycles

Using ***spp.mod***'s model as inspiration we created ***scpp.mod*** where, if we don't take special care for the source and target nodes then we will find the only paths that can satisfy the constraint are cycles since each node will have one outgoing edge and one incoming edge. However, in order to find all the cycles in the graph we must keep track of previous solutions and forbid then in their entirety but partially (some of the edges might be used in other paths). The way we achieved this behaviour was by subtracting the selected selected edges from the regular selected edges and guaranteeing that the difference is greater or equal than one, meaning that ***not all forbidden edges can be chosen and that at least one regular edge must be chosen*** (or else the shortest cycle would be to choose nothing).

Hence, the following model tries to do just that, where ***E*** represents the graph's edges and ***FE*** represents the forbidden edges of previous solution:

$$min \sum_{(i,j)\in E} c[i,j] * x[i,j]$$

$$s.t. : \sum_{(j,i)\in E} x[j,i] = \sum_{(i,j)\in E} x[i,j], \forall i = 0 \dots n$$

$$s.t : \sum_{(i,j)\in E} x[i,j] \geq 1 + \sum_{(i,j)\in FE} x[i,j];$$

We know that we have identified all solutions when the model can no longer output a feasible solution. This model correctly identified the following cycles:

- [1 2], [2 3], [3 1]
- [4 5], [5 7], [7 4]
- [4 5], [5 6], [6 7], [7 4]

3.3 All Paths

Having tried the previous approach to generate all acyclic paths from the source node to the target node and failed, we decide do instead use a more specific incremental restriction approach where we specify directly which paths are strictly forbidden from being repeated. Therefore, each of the following constraints forbids a specific path by enumerating which edges cannot appear simultaneously and exclusively.

- s.t. constraint1: (sum(i,j) in E x[i,j]) - (x[0,4] + x[4,8]) $\geq$ 1;
- s.t. constraint2: (sum(i,j) in E x[i,j]) - (x[0,1] + x[1,8]) $\geq$ 1;
- s.t. constraint3: (sum(i,j) in E x[i,j]) - (x[0,4] + x[1,2] + x[2,3] + x[3,1] + x[4,8]) $\geq$ 1;
- s.t. constraint4: (sum(i,j) in E x[i,j]) - (x[0,1] + x[1,8] + x[4,5] + x[5,7] + x[7,4]) $\geq$ 1;
- s.t. constraint5: (sum(i,j) in E x[i,j]) - (x[0,1] + x[1,8] + x[4,5] + x[5,6] + x[6,7] + x[7,4]) $\geq$ 1;

From the previously gathered constraints, we can see that the model found the following feasible solutions/paths:

- [0,4], [4,8]
- [0,1], [1,8]
- [0,4], [1,2], [2,3], [3,1], [4,8]
- [0,1], [1,8], [4,5], [5,7], [7,4]
- [0,1], [1,8], [4,5], [5,6], [6,7], [7,4]

As can be seen, the 3rd, 4th and 5th solution do, unfortunately, include cycles, but they are disconnected from the main path from the source node to the target node. This does not affect our results because the superfluous edges belonging to the cycle will be discarded in the post-processing.

This behaviour is due to the fact that these solutions do, in fact, have only 1 outgoing and 1 incoming edge for each node (with special care being given to the source and target nodes). This is an undesired side effect which results in far more redundant "unique" solutions being found (a combination of each acyclic and connect path with every cycle that shares no nodes with it).

3.4 All Paths with Cycles

Having determined the set of acyclic paths, basis edges and cycles for the graph we can now proceed to generating paths with cycles in a more intelligent manner than recursion.

For each acyclic path e combine all cycles that have at least one node in common with the path and, if there are no more than 1 occurrences of all basis edges in the resulting path it is deemed relevant for future path selection, otherwise the path is discarded.

In addition to this we also calculate the node, edge and edge-pair coverage of each relevant path, which is used in the partial coverage problem variant.

As a result we get the following relevant paths, numbered as such:

1. [0, 4, 8]
2. [0, 4, 5, 7, 4, 8]
3. [0, 4, 5, 6, 7, 4, 8]
4. [0, 4, 5, 6, 7, 4, 5, 7, 4, 8]
5. [0, 1, 8]
6. [0, 1, 2, 3, 1, 8]

This step is done entirely in python, namely in ***python/path_generation.py*** and as was described above.

This is the most time consuming step of the whole process due to the fact that the number of combinations of cycles to be included in each path can be overwhelming, for instance, if 5 cycles share a node with a path then 32 paths will be generated. A true combinatorial explosion.

4 Path-set selection

Using the knowledge covered in [1] for optimal path selection we implemented the following model

```
param n, integer, >= 0;
/* number of paths */

set I := {i in 1..n};
set J := {j in 1..maxC};

/* Path Coverage Frequency Matrix */
param f{i in I, j in J}, binary, >= 0;
```

```
/* Path Weight Matrix */
param w{i in I}, integer, >= 0;

/* Path Selection Matrix */
var x{I}, binary, >= 0;

/* Minimize with respect to the weight of each path */
minimize obj : sum{i in I} x[i] * w[i];

/* Selected Paths Coverage & with leat one is chosen */
s.t. pathCoverage{j in J}: sum{i in I} f[i,j] * x[i] >= 1;
```

which also takes into account the weight (number of nodes traversed) of each path and, sure enough, the results point out which paths we need to select in order to achieve total coverage for the provided graphs regarding each type of coverage (node, edge, edge-pair).

For our beloved ***cfg0_1.dat*** test case these were the following coverage outputs.
Node Coverage:

- [0, 4, 5, 6, 7, 4, 8]
- [0, 1, 2, 3, 1, 8]

Edge Coverage:

- [0, 4, 5, 6, 7, 4, 5, 7, 4, 8]
- [0, 1, 2, 3, 1, 8]

Edge-pair Coverage:

- [0, 4, 5, 6, 7, 4, 5, 7, 4, 8]
- [0, 1, 2, 3, 1, 8]

As expected, these results do in fact provide

5 Path-set cost/coverage trade-off

With the previous problem's solution in mind we simply needed to guarantee that the sum of satisfied columns ,or rather, covered nodes/edges/edge-pairs in the frequency matrix was at least the minimum desired coverage.

```
/* Max. Coverage */
param maxC, integer, >= 0;

/* Min Coverage Percent. */
param minCP, integer, >= 0;

param n, integer, >= 0;
```

```
/* number of paths */

set I := {i in 1..n};
set J := {j in 1..maxC};

/* Path Coverage Frequency Matrix */
param f{i in I, j in J}, binary, >= 0;

/* Path Weight Matrix */
param w{i in I}, integer, >= 0;

/* Path Selection Matrix */
var x{I}, binary, >= 0;

/* Checks if column (node, edge, edge-pair) is covered */
var t{J}, binary, >=0;

/* Minimize with respect to the weight of each path */
minimize obj : sum{i in I} x[i] * w[i];

/* Fill the auxiliary variable with whether each column is covered or not */
s.t. columnCover{j in J}: t[j] = sum{i in I} f[i,j] * x[i];

/* Cover at least Min. Coverage Percent. */
s.t. minPathCoverage: sum{j in J} t[j] * 100 >= minCP * maxC;
```

In order to test our model we decided to use 3 different minimum coverage criteria. Most of the outputs prove make sense (not just at first glance), except for the occasional lack of solutions (which is expanded upon later on).

5.1 75% Coverage

75% Node Coverage:

- ***LP HAS NO PRIMAL FEASIBLE SOLUTION***, in other words, no solution was found. This happened for higher minimum node coverage values (75%+) although we couldn't really understand why.

75% Edge Coverage:

- [0, 4, 5, 7, 4, 8]
- [0, 1, 2, 3, 1, 8]

75% Edge-pair Coverage:

- [0, 4, 5, 7, 4, 8]
- [0, 1, 2, 3, 1, 8]

5.2 50% Coverage

50% Node Coverage:

- [0, 4, 5, 7, 4, 8]

50% Edge Coverage:

- [0, 4, 5, 6, 7, 4, 8]

50% Edge-pair Coverage:

- [0, 4, 5, 6, 7, 4, 8]
- [0, 1, 8]

5.3 25% Coverage

25% Node Coverage:

- [0, 4, 8]

25% Edge Coverage:

- [0, 1, 2, 3, 1, 8]

25% Edge-pair Coverage:

- [0, 1, 2, 3, 1, 8]

References

[1] C.-G. Chung and J.-G. Lee, “An Enhanced Zero-One Optimal Path Set Selection Method,” Journal of Systems Software, vol. 39, pp. 145–164, 1997.