Testing Hierarchical Systems

Damon Mosk-Aoyama [*] Mihalis Yannakakis [†]

Abstract

We investigate the testing of hierarchical (modular) systems, in which individual modules are modeled by finite state machines. Given a hierarchical system, we are interested in finding a small set of tests that exercises all the transitions of the system. We present tight approximation algorithms and hardness results for the problem. Our techniques extend to other criteria and metrics.

1 Introduction

Testing of systems involves the use of a reference model that specifies how the system is supposed to behave. The model is used to generate an appropriate set of test cases which are applied to the system, and its response is compared to that prescribed by the model. The space of possible test cases is typically extremely large or infinite, so it is important to select the tests carefully to achieve a desired degree of fault coverage with the minimum use of resources.

For testing the behavioral aspects of systems, finite state machines, and their variants, are the most common model. There is extensive work over many years on this subject; see [12] for a survey. In order to specify complex systems, it is convenient (and often essential) to do this modularly in a hierarchical fashion. Hierarchical state machines are finite state machines whose states themselves can be other machines. The ability to build hierarchical models in this way is a central ingredient of specification formalisms such as statecharts [9], and object-oriented software design methodologies and languages such as OMT [14], ROOM [15], and the Unified Modeling Language (UML [6]), and related computer-aided software engineering tools. The hierarchical capability is useful also in formalisms and tools for requirements and testing. For instance, the ITU standard Z.120 (MSC'96) for message sequence charts (see [13]) formalizes scenarios of distributed systems in terms of hierarchical (high-level) message sequence charts (MSC's), which are basically hierarchical graphs built from basic MSC's (message exchanges). Lucent's *uBET* tool [10, 16] uses a similar model for the capture and testing of scenario requirements. A commercial testing tool, Testmaster [5] (originally by Teradyne, then Empirix), uses hierarchical extended finite state machines (FSM's extended with variables) as its reference model.

Hierarchical models were studied analytically in recent years with respect to their properties, expressive power, and algorithmic analysis and verification [2, 4, 3]. They offer exponential succinctness, arising from the ability to specify a module once and reuse it many times in different contexts; yet, they can in many cases be analyzed efficiently without paying a penalty for the succinctness.

In this paper we investigate the development of system tests from hierarchical reference models. One way to generate tests from a given hierarchical model (e.g. a hierarchical state machine) is to flatten the hierarchy by recursively substituting the lower-level modules in the upper ones, thereby obtaining a (larger) ordinary FSM, and then generate tests from that flat model. This is for example the approach followed in uBET: the hierarchical MSC is flattened and then a set of test sequences is generated which covers all the edges of the flat graph, and minimizes (i) the number of sequences, and then (ii) the total length of the sequences among all sets that minimize (i); such an optimal set of test sequences can be found in polynomial time for flat graphs with flow techniques. This approach has potentially two disadvantages in general: first, the flattened graph can be much larger than the input hierarchical model, namely it can have size n^d where n is the number of nodes of a module and d the depth of the hierarchy (the number of levels); second, this method covers every edge of each module in every possible context and can thus result in general in a very large number of tests. Especially in the context of system testing, the tests may be run physically on the actual equipment, which means that setting up and running a test takes actual real time, and one can only run a limited number of tests.

In this paper we investigate the optimal test generation problem under the standard transition coverage cri-

[*]Department of Computer Science, Stanford University, damonma@cs.stanford.edu. Supported in part by a Stanford School of Engineering fellowship.

[†]Department of Computer Science, Columbia University, mihalis@cs.columbia.edu. Supported in part by an NSF Grant.

terion: given a hierarchical model, compute a minimum number of tests such that every transition of every module is included in at least one test; if a module is reused many times in many different contexts, we may choose to cover some of its transitions in one place, and other transitions in another place. We show that the problem is NP-hard in the non-flat case – i.e., from depth 2 up. We give an approximation algorithm that achieves ratio d, equal to the depth of the hierarchy. Conversely, we show that for every d, it is NP-hard to do better than ratio d. For the special case in which the hierarchy is a tree, we give an improved algorithm whose approximation ratio is the rank of the tree. The rank is always no larger than the depth, but it can be much smaller; it is always at most logarithmic in the number of modules in the system. We apply our techniques also to the minimization of the lengths of the tests, and more generally minimization when the transitions have given costs, and we give a $2d-1$ approximation for this setting. We also address several other extensions.

The remainder of the paper is organized as follows. In Section 2, we give the basic definitions and a hierarchical flow representation of the problem and a solution. Section 3 studies the special case of the minimum complete test set problem in which the hierarchy is a tree. In Section 4, we relax this restriction of the hierarchy, and present an approximation algorithm for the minimum complete test set problem in general hierarchies. Section 5 considers the hardness of the problem. In Section 6 we extend the algorithms to the state coverage criterion, and variants of the model, and in Section 7 we address general edge costs. Finally, Section 8 presents brief concluding remarks and further directions.

2 Preliminaries

A *flat model* (or *testing graph*) is a directed graph $G = (V, E)$, with a unique source vertex $s \in V$ of in-degree 0, (called the *initial* or *entry* vertex), and a unique sink vertex $t \in V$ of out-degree 0, called the *final* or *exit* vertex; all other vertices are assumed to be reachable from s and can reach t. In general, we will write $V(G)$, $E(G)$, $s(G)$, and $t(G)$ to denote the vertex set, edge set, source vertex, and sink vertex, respectively, of a particular graph G.

A *hierarchical model (graph)* H is a tuple $(M_1, \ldots, M_K)$ of *modules*, where each module M_i has the following components: (1) a set of vertices $V_i = N_i \cup B_i$, which is partitioned into a set N_i of (ordinary) nodes and a set B_i of *supernodes* (or *boxes*), (2) a set of directed edges E_i between the vertices, (3) an initial source node $s_i \in N_i$, (4) a final sink node $t_i \in N_i$, (5) an indexing function Y_i that maps every supernode of M_i to a module M_j with a higher index $j > i$. Intuitively, if $Y_i(u) = M_j$ for a supernode u of module M_i, then this indicates that u refers to an instance (a copy) of the module M_j. Edges entering and leaving u implicitly connect to the initial and final nodes of the copy of M_j. It is assumed (wlog) that all vertices of each module M_i are reachable from the initial node s_i and can reach the final node t_i. Figure 1(a) shows an example of a hierarchical model with three modules M_1, M_2, M_3. Note that many supernodes of a module (or of different modules) can map to the same module; this is the essence of reuse: defining a module once and reusing it many times.

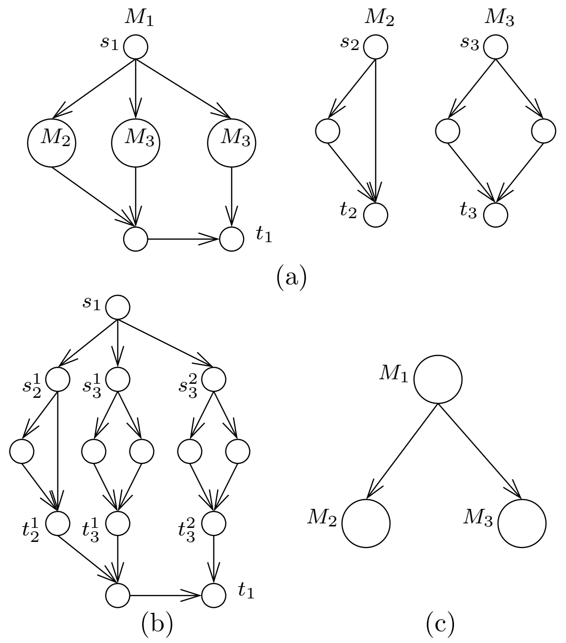

Figure 1: (a) A collection of modules that comprise a hierarchical model. (b) The expanded graph for the system. (c) The containment graph for the system.

Each module M_i of the hierarchical model can be viewed as a succinct representation of a flat model $flat(M_i)$, which is obtained by recursively replacing all supernodes with their corresponding modules and repeating the process until there are no supernodes left. We refer to $flat(M_i)$ as the *flat* or *expanded* graph of M_i. The expanded graph of the whole system is $flat(M_1)$. Figure 1(b) shows the expanded graph for the system in Figure 1(a). In general, the size of the expanded graph may be exponential in the size of the hierarchical model (i.e., the total number of vertices and edges of all the modules).

If module M_i has a supernode that is mapped to module M_j, then we say that M_i *contains* M_j. Define a directed graph, called the *containment* or *hierarchy* graph, whose vertices correspond to the modules, and which has an edge from M_i to M_j iff module M_i contains module M_j. Since every supernode of M_i is mapped to a module with higher index, the containment graph is acyclic, and a topological order of it is given by the indexes of the modules. The first (top-level) module M_1 represents the whole system. We may assume without loss of generality that every module is reachable in the containment graph from the top-level module. The containment graph for the system in Figure 1(a) is presented in Figure 1(c).

Define the *depth* of a module M, written $d(M)$, as the number of vertices in the longest path that starts at M in the containment graph. The depth of a containment graph is defined to be the depth of the top-level module. If each module has at most n vertices and the depth of the top module is d, then the expanded flat graph has $O(n^d)$ nodes. In practice, d is not very large (e.g., $d \leq 10$), but it is not small enough to regard n^d as "efficient" (polynomial).

If the containment graph is a tree (rooted at the top module), we can define the *rank* of each node of the tree and the corresponding module M, written $r(M)$, as follows. If M is a leaf, then $r(M) = 1$. Otherwise, suppose that the node (module) M has k children in the tree, $M_{i_1}, \ldots, M_{i_k}$. Compute the rank for each child M_{i_j}, and order the children by descending rank, so that $r(M_{i_1}) \geq r(M_{i_2}) \geq \cdots \geq r(M_{i_k})$. Then, the rank of M is $r(M_{i_1})$ if $k = 1$, and $r(M) = \max\{r(M_{i_1}), r(M_{i_2})+1\}$ if $k > 1$. We define the rank of a containment tree to be the rank of the top-level module, which is the root of the tree.

Hierarchical graphs form the basis for representing various types of models such as *hierarchical state machines* (nodes correspond to states) and *hierarchical (high-level) message sequence charts* (nodes correspond to basic MSC's).

A *test* of a hierarchical model H with top module M_1 is defined to be a path of $flat(M_1)$ from the initial to the final node. Note that a test traverses a set of edges of the top module M_1 as well as (possibly) various edges of submodules; we say that the test *covers* all the edges that it traverses. A test represents a complete execution of the system.

During the process of testing a system, we seek a set of tests of a particular fault coverage, which will provide evidence that the system conforms to the specification. The standard of fault coverage that we employ in this paper is the basic *transition coverage* criterion: we define a *complete test set* of a hierarchical model H as a collection of tests such that every edge in every module is covered by at least one test in the collection. Note that if a particular module occurs at several different positions in a system, then its transitions will appear multiple times in the expanded graph for the system. A complete test set, however, need contain only one occurrence of each transition in any such module. Our primary goal here is to minimize the number of tests in our test set. The *minimum complete test set problem* is the problem of finding a complete test set for a given hierarchical model that contains the minimum number of tests. We will address also the minimization of the length of the tests and a more general cost metric.

Observe that within any strongly connected component in a module, it is always possible to construct a single path that contains all the edges in that component. For this reason, if we wish to minimize the number of tests, we may shrink all the strong components as a preprocessing step to simplify subsequent computations. For length minimization we do not shrink components.

2.1 A Network Flow Representation of Test Sets Consider a hierarchical model H with top-level module M_1. We will represent a test set of H compactly as a collection of integral flows in the modules of H. Consider a test set S, i.e. a set of source-to-sink paths in $flat(M_1)$. This induces a (integral) flow f on $flat(M_1)$, where the flow on each edge is the number of paths of S that traverse the edge. The value of the flow f, denoted $|f|$, is equal to the number of tests. The test set induces also a flow f^M on each module M of the hierarchical model: the flow on each edge e of M is equal to the sum of the flows of all the edges of $flat(M)$ that correspond to the edge e of M; the test set covers the edge iff the flow on e is at least 1. Note that each f^M is indeed a valid flow, i.e. satisfies flow conservation at all vertices of M except the source and the sink. This collection of flows $\{f^M\}$ satisfies also the following *(strong) coupling property*: for each module M, the value $|f^M|$ of the flow f^M is equal to the sum of the flows through all the supernodes u of all the modules such that u is mapped to M. We call a flow collection $f = \{f^M\}$ *feasible* if it satisfies the edge coverage and the coupling properties. Its value $|f|$ is the value $|f^{M_1}|$ of the flow in the top-level module.

Conversely, consider a collection of integral flows $\{f^M\}$ which satisfies (i) the edge coverage property, and (ii) the *weak coupling property*: for each module M, the total amount of all the flows through all the supernodes that are mapped to M is at least the value $|f^M|$ of the flow for M. Then we claim that we can derive a test set whose size is equal to the value of the flow in the top module M_1. We sketch the argument. First, if property (ii) is satisfied with a strict inequality for some module M (note that M is not the top-level module), then we can augment the flow for M by adding an arbitrary set of source-sink paths in M so that (ii) is satisfied with equality. We can do this augmentation in a topological order top-down. The value of the flow at the top module M_1 remains the same. Now decompose the flow in each module into a set of source-sink paths. Since (ii) is satisfied with equality, we can associate each unit of flow through each supernode u with a path for the module M to which u is mapped. We can then expand inductively each path of each module M_i to a path of $flat(M_i)$ by replacing each supernode by the associated path from the corresponding lower-level module. Thus, the paths of the top module M_1 yield a complete test set.

We summarize the above discussion in the following.

Proposition 2.1. *Every feasible integer flow collection $f = \{f^M\}$ corresponds to a complete test set of size $|f|$, and vice versa. Every integer flow collection that satisfies the edge coverage and weak coupling properties can be transformed to a feasible flow collection with the same value.*

Our algorithms will compute a test set in the form of a feasible flow $f = \{f^M\}$ for the modules. Observe that this is a succinct representation of a solution, linear in the total size of the hierarchical model. Note, however, that if we expand the flows to paths in the flat graph, some of the paths may end up being exponential in the depth d (i.e. of length n^d). Although this is pathological, in general this may in fact be unavoidable, i.e. it is possible that some edges of some modules can be covered only by paths in the flat graph that are this long. Our algorithms will compute approximately optimal solutions without constructing explicitly the flat paths, and without incurring the n^d time penalty.

In the flow formulations, it will be sometimes convenient to introduce an edge $(t(M), s(M))$ in a module M from the sink to the source. Then, given a circulation in M, we can remove the edge $(t(M), s(M))$ to obtain a flow in M from the source to the sink.

3 Algorithm for Tree Hierarchies

After presenting a fractional relaxation involving a set of LP's, we give an algorithm whose approximation ratio is at most the rank of the hierarchy (the containment tree).

3.1 Fractional Relaxations For any module G, let $\mathrm{OPT}(G)$ denote the number of tests in a minimum complete test set for G. If G has no children in the containment tree, then we can compute $\mathrm{OPT}(G)$ by solving the Chinese postman problem, which can be done in polynomial time in directed graphs [8]. In this case, we can obtain a solution by solving the following minimum cost circulation problem (MC), with lower bounds $\ell_e = 1$ for all $e \in E(G)$. For a vertex $v \in V(G)$, we let $A^+(v)$ $(A^-(v))$ denote the set of outgoing (incoming) edges from (to) v.

$$\text{(MC)}\quad \begin{aligned} &\text{minimize } f_{(t(G),s(G))} \\ &\text{subject to} \\ &\sum_{e\in A^+(v)} f_e - \sum_{e\in A^-(v)} f_e = 0, && \forall v \in V(G) \\ &f_e \ge \ell_e, && \forall e \in E(G) \end{aligned}$$

Since all the lower and upper bounds are integers, the optimal solution will be integral [1]. Thus, for a module G with depth $d(G) = 1$ in the containment tree, the LP (MC) has an optimal solution that is integral.

For modules G of higher depth, we set up and solve an LP whose optimal value $\mathrm{OPT}_f(G)$ provides a lower bound on $\mathrm{OPT}(G)$. We do this bottom-up in the hierarchy. Suppose that G has children $G_1, \ldots, G_k$, and let $U(G, G_i)$ denote the set of supernodes of G that are mapped to G_i. We set up and solve the following LP (CF), where $\mathrm{OPT}_f(G_i) = \mathrm{OPT}(G_i)$ if G_i has depth 1.

$$\text{(CF)}\quad \begin{aligned} &\text{minimize } x_{(t(G),s(G))} \\ &\text{subject to} \\ &\sum_{e\in A^+(v)} x_e - \sum_{e\in A^-(v)} x_e = 0, && \forall v \in V(G) \\ &\sum_{u\in U(G,G_i)} \sum_{e\in A^-(u)} x_e \ge \lceil \mathrm{OPT}_f(G_i) \rceil, && i = 1, \ldots, k \\ &x_e \ge 1, && \forall e \in E(G) \end{aligned}$$

Let COMPUTE-FRACTIONAL-FLOW be the procedure that computes bottom-up the $\mathrm{OPT}_f(G)$ values for all modules G. It is easy to show inductively that for every module G, and in particular for the top module, the minimum (integral) complete test set for G will be feasible for the LP (CF) for G, and therefore $\lceil \mathrm{OPT}_f(G) \rceil$ will be a lower bound on $\mathrm{OPT}(G)$. The integrality gap between the two quantities is approximately d, the depth of the containment graph.

Theorem 3.1. *For any $k > 0$ and $d > 0$, there is an instance of the minimum complete test set problem consisting of a module L^d at the root of a containment tree of depth $d(L^d) = d$, such that $\mathrm{OPT}_f(L^d) \le (k + d)\left(\frac{k+1}{k}\right)^{d-1}$, and $\mathrm{OPT}(L^d) \ge (k+1)d$.*

We omit the proof from this extended abstract. Theorem 3.1 implies that the ratio between $\mathrm{OPT}(L^d)$ and $\mathrm{OPT}_f(L^d)$ is at least $\frac{d}{\left(\frac{k+d}{k+1}\right)\left(\frac{k+1}{k}\right)^{d-1}}$ for all $k > 0$; this quantity approaches d as $k \to \infty$ for any fixed $d > 0$.

3.2 Rank Approximation Ratio We will present an algorithm whose ratio is bounded by the rank of the containment tree. The rank of a tree is always upper bounded by the depth. In the other direction, the rank can be arbitrarily smaller than the depth. For example, if the tree is just a path of d nodes, then the depth is d but the rank is 1; in this case we can compute an optimal solution as we shall show. Of course, in the worst case, the rank can be equal to the depth, for example for a complete binary tree.

The following subroutine is used in this and our other algorithms. The subroutine takes as input a module G, a (fractional) flow x and a positive integer b, and returns an integral flow.

```
ROUND($G, x, b$)
  1. Construct an instance of (MC) in $G$ with a lower
     bound $\ell_e = \lceil (b-1)x_e \rceil$ for each edge $e \in E(G)$.
     Solve this instance to obtain a source-sink flow $f$.
  2. Output the flow $f$.
```

LEMMA 3.1. *Given a module G, a flow x in G that covers all the edges, and an integer $b > 1$,* ROUND *produces an integral flow f in G that covers all the edges such that $|f| \leq b|x|$.*

The overall structure of our algorithm, which we will call Algorithm TR, is to first find the optimal fractional flows as in the previous subsection, and then process the modules a second time bottom-up in the tree to construct integral solutions. Consider a module G with children $G_1, \ldots, G_k$, and assume that the children are indexed by descending rank. Recall that the rank of G is defined as $r(G) = \max\{r(G_1), r(G_2) + 1\}$. It follows from this definition that $r(G) = r(G_1)$ or $r(G) = r(G_1) + 1$.

If $r(G) = r(G_1) + 1$, then the rank increases by 1 from G_1 to G. In this case the algorithm uses the ROUND subroutine.

If $r(G) = r(G_1)$, then we use a different procedure. Note that in this case, all the children $G_2, \ldots, G_k$ other than G_1 have ranks that are less than the rank of G. Algorithm TR uses an optimal solution x to the linear program (CF) for G, and then solves the following linear program (CR).

$$\text{(CR)} \quad \text{minimize } y_{(t(G),s(G))}$$
$$\text{subject to}$$
$$\sum_{e \in A^+(v)} y_e - \sum_{e \in A^-(v)} y_e = 0, \qquad \forall v \in V(G)$$
$$(3.1) \quad \sum_{u \in U(G,G_1)} \sum_{e \in A^-(u)} y_e \geq r(G) \lceil \mathrm{OPT}_f(G_1) \rceil$$
$$(3.2) \quad y_e \geq \max\{\lceil (r(G)-1)x_e \rceil, 1\}, \qquad \forall e \in E(G)$$

The following property of an optimal solution to (CR) is analogous to that in Lemma 3.1, and the two lemmas may be proved using similar arguments.

LEMMA 3.2. *For any module G, if x is a feasible solution to the linear program (CF) for G, then the value of an optimal solution y to the linear program (CR) corresponding to x for G satisfies $|y| \leq r(G)|x|$.*

The optimal solution y may be fractional. For a module G and a flow y in G, we define the *fractional graph G_y* to be the undirected graph on the vertex set $V(G)$ that has an edge (u, v) if and only if y_e is not an integer for $e = (u, v)$ or $e = (v, u)$, where $e \neq (t(G), s(G))$. Algorithm TR uses the following key lemma to obtain an integral flow from the fractional flow x.

LEMMA 3.3. *For any module G, if x is a feasible solution to the linear program (CF) for G, then the linear program (CR) corresponding to x for G has an optimal solution y such that the fractional graph G_y is acyclic. Furthermore, such a solution y can be derived efficiently from any optimal solution y' of (CR).*

The proof is constructive and involves an algorithm that takes a solution y' and iteratively removes cycles from its associated graph $G_{y'}$, modifying the solution y' in the process without making it worse, until it finally arrives at a solution y with an acyclic G_y. The details are given in the Appendix.

Lemma 3.3 provides us with a method for finding an integral solution to the linear program (CR) of value $\lceil |y'| \rceil$, where y' is an optimal solution to (CR). First, we use the algorithm described in the proof of the lemma to obtain an optimal solution y to (CR) such that the fractional graph G_y is acyclic. Because the flow in y is conserved at all vertices other than the source and the sink, and the fractional graph G_y is acyclic, if there are edges in the flow with fractional amounts of flow, then these edges must form a path in G_y from the source to the sink. It is possible to modify the flow on the edges along this path to obtain a feasible integer flow of value $\lceil |y'| \rceil$.

The following is a description of Algorithm TR, called on the hierarchical model H.

```
Algorithm TR($H$)
  1. Execute COMPUTE-FRACTIONAL-FLOW to
     obtain optimal (fractional) solutions $x^G$ to the LP
     (CF) for all modules $G$ in the containment tree
     such that $d(G) > 1$, and integral source-sink flows
     $f^G$ for all modules $G$ with $d(G) = 1$.
  2. For each module $G$ in the containment tree such
     that $d(G) > 1$, let $G_1, \ldots, G_k$ be the children of $G$
```

in the containment tree, ordered by descending rank. If $r(G) > r(G_1)$, then execute ROUND$(G, x^G, r(G))$ to obtain a source-sink flow f^G in G.

3. In the case that $r(G) = r(G_1)$, construct the linear program (CR) corresponding to x^G for G. Compute an optimal solution y' to (CR).
4. Use the iterative cycle-canceling procedure of Lemma 3.3 to convert y' to an optimal solution y to (CR) such that the fractional graph G_y is acyclic. If the fractional graph G_y has any edges, then modify the flow on the source-sink path in G_y to obtain an integral source-sink flow f^G in G.
5. Transform the flows f^G to a feasible flow collection as in Section 2.1, and output the resulting test set.

We can show that the approximation ratio of TR is at most the rank of the containment tree. The proof is given in the Appendix.

Theorem 3.2. *Given a hierachical model H for which the containment graph is a tree rooted at the top-level module M_1, Algorithm TR finds a complete test set for H of size at most $r(M_1)\mathrm{OPT}(M_1)$.*

A consequence of the definition of the rank of a tree is that the rank of a tree on n vertices is $O(\log n)$. The approximation ratio of Algorithm TR is therefore $O(\log K)$, where K is the number of modules in the system, which is the number of vertices in the containment tree.

4 General Hierarchies

Consider a hierarchical model H with modules $M_1, \ldots, M_K$ in topological order of the hierarchy (the containment graph), i.e., if a module M_i contains a supernode that maps to module M_j, then $i < j$. The top-level module is M_1.

Our algorithm begins by finding optimal fractional solutions to the minimum complete test set problem in all the modules. A difference with the tree case is that we cannot do this by solving a sequence of separate LP's, one for each module. Rather, we solve a single linear program for all the modules together; this is generally more costly (but still polynomial in the input of course). This LP has a variable x_e for each edge $e \in E(M_i)$ for all $i = 1, \ldots, K$, which represents the amount of flow on the edge e. In addition, for each pair of modules M_i, M_j such that M_i contains M_j, the LP, which we present below and refer to as (CD), has a variable z_{ij} that represents the amount of flow in M_i that goes into supernodes that are mapped to M_j.

$$\text{(CD)} \quad \text{minimize } x_{(t(M_1),s(M_1))}$$
$$\text{subject to}$$
$$\sum_{e\in A^+(v)} x_e - \sum_{e\in A^-(v)} x_e = 0, \quad \forall v \in \bigcup_{i=1}^{K} V(M_i)$$
$$(4.3) \quad z_{ij} = \sum_{u\in U(M_i,M_j)} \sum_{e\in A^-(u)} x_e, \quad \begin{array}{l} i = 1, \ldots, K-1, \\ j = i+1, \ldots, K \end{array}$$
$$(4.4) \quad \sum_{i=1}^{j-1} z_{ij} \geq x_{(t(M_j),s(M_j))}, \quad j = 2, \ldots, K$$
$$x_e \geq 1, \quad \forall e \in \bigcup_{i=1}^{K} E(M_i)$$

For any $i \in \{1, \ldots, K\}$, let x^{M_i} denote the flow in M_i obtained by restricting x to those edges in M_i, so that $x_e^{M_i} = x_e$ for all $e \in E(M_i)$. We now outline the steps performed by our algorithm for the minimum complete test set problem in arbitrary containment graphs, which we will refer to as Algorithm D, when it is called on the hierachical model H.

Algorithm D(H)

1. Construct the linear program (CD). Compute an optimal solution (x, z) to (CD).
2. For each module M_i in the containment graph such that $d(M_i) = 1$, solve the minimum cost circulation instance (MC) in M_i with lower bounds $\ell_e = 1$ for all $e \in E(M_i)$ to obtain an integral source-sink flow f^{M_i} in M_i.
3. For each module M_i such that $d(M_i) > 1$, execute ROUND$(M_i, x^{M_i}, d(M_i))$ to obtain a source-sink flow f^{M_i} in M_i.
4. Transform the set of flows f^{M_i} as in Section 2.1 to a feasible flow collection, and output the resulting test set.

Let OPT_f denote the optimal value of the LP (CD), and let OPT denote the size of the minimum complete test set for the system. Since any complete test set is a feasible solution to (CD), $\mathrm{OPT}_f \leq \mathrm{OPT}$. The following theorem establishes an upper bound on the approximation ratio of Algorithm D.

Theorem 4.1. *Given a hierachical model H with a top-level module M_1, Algorithm D finds a complete test set for H of size at most $d(M_1)\mathrm{OPT}$.*

The proof is given in the Appendix.

5 Hardness of Approximation

We show that the problem of finding a minimum complete test set is NP-hard, even for depth 2. Furthermore, we show an inapproximability lower bound that matches our upper bound.

Theorem 5.1. *For every depth d, the problem of computing a minimum complete test set for hierarchical models of depth d cannot be approximated to a factor smaller than d unless* P = NP. *This holds even if the containment graph is restricted to be a tree.*

Our reductions are from the *Vertex Cover* problem in k-uniform hypergraphs. It was shown in [7] that for any $k \geq 3$ and $\epsilon > 0$, it is NP-hard to tell for a given hypergraph H with n vertices whether the value of the optimal vertex cover $\mathrm{OPT}(H)$ is less than $\frac{1}{k-1-\epsilon}n$, or more than $(1-\epsilon)n$.

Our reduction is recursive and encodes in the modules at each level of the hierarchy the given hard hypergraph instance H, so that the inapproximability ratio increases by 1 with each level. For $d = 2$, we construct from a k-uniform hypergraph H with n vertices a depth-2 model G so that its minimum complete test set has size $\mathrm{OPT}(G) = n + \mathrm{OPT}(H)$; hence it is NP-hard to tell whether $\mathrm{OPT}(G) \leq n(1+\frac{1}{k-1-\epsilon})$ or $\mathrm{OPT}(G) > n(2-\epsilon)$. For general depth $d \geq 3$, we construct from H a suitable top module G and map its supernodes to inductively constructed hard models of depth $d-1$. The resulting hierarchical model has the property that it is NP-hard to tell whether $OPT(G) \leq n(1 + (d-1)/(k-1-\epsilon))$, or $OPT(G) > dn(1-\epsilon)$. The ratio goes to d as $k \to \infty$ and $\epsilon \to 0$. We omit the details of the construction due to space limitations.

6 Extensions

Our results extend to several variants of the model and other coverage criteria.

State Coverage: In this criterion we want to find a minimum set of tests that covers all vertices (nodes and supernodes) of all the modules. Both the rank algorithm and the depth algorithm can be adapted easily to handle this criterion. Theorems 3.2 and 4.1 remain valid.

Multiple Final States: Suppose that modules can have many final states (nodes), at which the execution in a module may terminate, transferring control back to the higher-level module. That is, when an execution of the system (a path) in module M reaches a supernode v mapped to module N, it follows a path in N from the initial state of N to one of its final states, and then the path resumes in the higher-level module M following an edge out of v. Again, both the rank algorithm and the depth algorithm can be extended to this model.

Multiple Visible Exits: In the above model, when execution finishes in module N and resumes at the higher-level module M, the continuation is independent of the final state. A more expressive model is to allow multiple exit states that are externally visible. In this model, the edges coming out of a supernode mapped to model N specify which particular exit of the module N they are connected to. The depth (but not the rank) algorithm can be extended to this model.

We summarize the results in the following theorem.

Theorem 6.1. *The problem of generating the minimum test set that covers all the states or all the transitions of a hierarchical model with multiple final states or multiple exits, can be approximated within a factor d, the depth of the hierarchy. If the hierarchy is a tree, then the problems can be approximated within the rank of the hierarchy, even if there are multiple final states, if they are not externally visible.*

Finally, we note one extension that cannot be approximated. Suppose that we only need to cover a given subset of required transitions. For flat models this makes no difference: we can compute a minimum set of tests that exercises the required transitions using essentially the same minimum flow algorithm. The problem is much harder for hierarchical models, even for depth 2. Although we could extend the depth and rank algorithms, they will not guarantee a constant factor approximation in this case. This is not surprising, as the problem for depth 2 models includes (via a reduction) as a (very) special case the Set Cover problem, and also the more general *Colored graph covering* problem [11], for which no logarithmic approximation is known.

Theorem 6.2. *The problem of finding a minimum set of tests that exercises a specified set of required transitions of a given depth-2 hierarchical model cannot be approximated within any constant factor unless* P = NP, *and cannot be approximated within $c \ln n$, $c < 1$, where n is the number of vertices of the model, unless* NP *is contained in* $\mathrm{DTIME}(n^{\log\log n})$.

7 Length Metric and Edge Costs

Suppose that every edge e has a given nonnegative cost c_e. The cost of a path is the sum of the costs of its edges. We are given a hierarchical model and we wish to find a set of tests that cover all the transitions of the model and which minimizes the total cost of the tests.

Minimization of the total length of the tests is the special case where $c_e = 1$ for all edges e. Minimization of the number of tests can also be expressed as a special case by letting $c_e = 1$ for all edges e coming out of the initial state of the top module, and $c_e = 0$ for all other edges. Other values of the costs can be used to express the relative importance ascribed to the number of tests versus their length, and to express the different times that it takes to execute different transitions (note that the transitions of testing models typically express a sequence of operations).

We can modify the depth algorithm to this setting and obtain an approximation with factor $2d-1$. This is not quite immediate and requires some care. Due to the space limitations, details are deferred to the full paper.

THEOREM 7.1. *For all d, the minimum cost complete test set problem for hierarchical models of depth d can be approximated in polynomial time with ratio $2d-1$.*

A similar algorithm applies also to the state coverage problem with the same $2d-1$ ratio, if the modules of the hierarchical model are acyclic. Note that even for flat cyclic graphs, the minimum cost state coverage problem is NP-hard and it is unknown if it has a constant factor approximation: the problem is equivalent to the asymmetric traveling salesman problem, for which the best ratio currently known is $\log n$.

Similar techniques apply to the case of multiple final states or multiple exits (including cyclic models for transition coverage); the ratio in the former case is $2d-1$, and in the latter case our current ratio is $3d$.

8 Conclusions

In this work, we studied the testing of hierarchical systems under the transition coverage criterion, and developed tight upper and lower bounds for the problem and several extensions. This work is the first rigorous study of the optimal testing of hierarchical systems. There is a wealth of open problems and further directions that deserve investigation, such as the extension to other types of coverage criteria that have been studied for flat models, investigation of hierarchical models with variables (extended hierarchical state machines), and a number of others.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall, 1993.

[2] R. Alur, S. Kannan, and M. Yannakakis. Communicating hierarchical state machines. In *Proc. 26th Intl. Coll. on Automata, Languages and Programming*, 1999.

[3] R. Alur and M. Yannakakis. Model checking of message sequence charts. In *Proc. 10th Intl. Conf. on Concurrency Theory*, pages 114–129. Springer-Verlag LNCS 1664, 1999.

[4] R. Alur and M. Yannakakis. Model checking of hierarchical state machines. *ACM Trans. on Progr. Languages and Systems*, 23:273–303, 2001.

[5] L. Apfelbaum. Automated functional test generation. In *Proc. IEEE Autotestcon Conference*, 1995.

[6] G. Booch, I. Jacobson, and J. Rumbaugh. *Unified Modeling Language User Guide.* Addison-Wesley, 1998.

[7] I. Dinur, V. Guruswami, S. Khot, and O. Regev. A new multilayered PCP and the hardness of hypergraph vertex cover. In *Proc. of the 35th ACM STOC*, pp. 595–601, 2003.

[8] J. Edmonds and E. L. Johnson. Matching, Euler tours and the Chinese postman. *Mathematical Programming*, 5:88–124, 1973.

[9] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.

[10] G. Holzmann, D. A. Peled, and M. H. Redberg. Design tools for requirements engineering. *Bell Labs Technical Journal*, 2:86–95, 1997.

[11] D. Lee and M. Yannakakis. Optimization problems from feature testing of communication protocols. In *Proc. of 4th International Conference on Network Protocols*, pages 66–75, 1996.

[12] D. Lee and M. Yannakakis. Principles and methods of testing finite state machines – a survey. *Proceedings of the IEEE*, 84:1090–1126, 1996.

[13] E. Rudolph, J. Grabowski, and P. Graubmann. Tutorial on message sequence charts. *Computer Networks and ISDN Systems*, 28:1629–1641, 1996.

[14] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-oriented modeling and design.* Prentice Hall, 1991.

[15] B. Selic, G. Gullekson, and P. T. Ward. *Real-time object oriented modeling and design.* J. Wiley, 1994.

[16] uBET. http://cm.bell-labs.com/cm/cs/what/ubet.

9 Appendix

Proof of Lemma 3.3

We give a constructive proof, exhibiting an algorithm for converting an arbitrary optimal solution y' to (CR) to an optimal solution y for which the fractional graph G_y is acyclic. Suppose that $G_{y'}$ has a cycle C', and fix some ordering $v_1, v_2, \ldots, v_q, v_1$ of the vertices on C'. Define a set of edges $C \subseteq E(G)$ as follows. Consider any pair (u, v) of consecutive vertices in this order on the cycle, so that $u = v_i$ and $v = v_{i+1}$ for some $i \in \{1, \ldots, q-1\}$, or $u = v_q$ and $v = v_1$. If $(u, v) \in E(G)$, then $(u, v) \in C$, and we refer to the edge (u, v) as an *ascending* edge. If $(v, u) \in E(G)$, then $(v, u) \in C$, and the edge (v, u) is called a *descending* edge. Because G is acyclic (we assume that all the strongly connected components of all the modules have been collapsed to nodes as a preprocessing step), for any edge $(u, v) \in C'$, exactly one of the directed edges (u, v)

and (v, u) is in $E(G)$, and therefore $|C| = |C'|$. Figure 2 shows an example of ascending and descending edges arising from a cycle in a fractional graph.

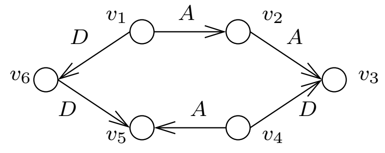

Figure 2: Edges in a module corresponding to a cycle in a fractional graph. Under the ordering of vertices shown, the edges marked with A are ascending edges, and the edges marked with D are descending edges.

Our cycle-removal algorithm will eliminate the cycle C' from the fractional graph $G_{y'}$ by changing the flow on the edges in C in y'. The algorithm ensures that some edge on C' is removed from the fractional graph when it sends flow around the cycle. Define two circulations c^1 and c^2 in G as follows. For each edge $e \in E(G) - C$, $c^1_e = c^2_e = 0$. Consider an edge $e \in C$, and define the following quantities α^1_e and α^2_e. If e is an ascending edge, then $\alpha^1_e = \lceil y'_e \rceil - y'_e$ and $\alpha^2_e = y'_e - \lfloor y'_e \rfloor$. On the other hand, if e is a descending edge, then $\alpha^1_e = y'_e - \lfloor y'_e \rfloor$ and $\alpha^2_e = \lceil y'_e \rceil - y'_e$. Let $\alpha^1 = \min_{e \in C}\{\alpha^1_e\} > 0$ and $\alpha^2 = \min_{e \in C}\{\alpha^2_e\} > 0$. For ascending edges $e \in C$, set $c^1_e = \alpha^1$ and $c^2_e = -\alpha^2$, and for descending edges $e \in C$, set $c^1_e = -\alpha^1$ and $c^2_e = \alpha^2$.

Compute the two flows $y^1 = y' + c^1$ and $y^2 = y' + c^2$. It follows from the definition of the circulations c^1 and c^2 that for all edges $e \in E(G)$, and for $i \in \{1, 2\}$, $y^i_e \geq \lfloor y'_e \rfloor \geq \max\{\lceil (r(G) - 1)x_e \rceil, 1\}$. This shows that the flows y^1 and y^2 both satisfy all the constraints (3.2) in the LP (CR).

Now, consider the constraint (3.1). For $i \in \{1, 2\}$, the following inequality results from the definition of the circulations c^1 and c^2.

$$\sum_{u \in U(G,G_1)} \sum_{e \in A^-(u)} y^i_e = \sum_{u \in U(G,G_1)} \sum_{e \in A^-(u)} (y'_e + c^i_e)$$

$$\geq r(G)\lceil \mathrm{OPT}_f(G_1) \rceil + \sum_{u \in U(G,G_1)} \sum_{e \in A^-(u) \cap C} c^i_e \tag{9.5}$$

Let C^A (C^D) be the set of ascending (descending) edges in C that enter supernodes that map to G_1 in G. We use these sets to compute $\sum_{u \in U(G,G_1)} \sum_{e \in A^-(u) \cap C} c^i_e$ for $i \in \{1, 2\}$.

$$\sum_{u \in U(G,G_1)} \sum_{e \in A^-(u) \cap C} c^1_e = \sum_{e \in C^A} c^1_e + \sum_{e \in C^D} c^1_e$$
$$= \sum_{e \in C^A} \alpha^1 + \sum_{e \in C^D} (-\alpha^1) = \alpha^1(|C^A| - |C^D|)$$

By a similar calculation, we obtain $\sum_{u \in U(G,G_1)} \sum_{e \in A^-(u) \cap C} c^2_e = \alpha^2(|C^D| - |C^A|)$. As $\alpha^1, \alpha^2 > 0$, it follows that for some $j \in \{1, 2\}$, $\sum_{u \in U(G,G_1)} \sum_{e \in A^-(u) \cap C} c^j_e \geq 0$. The inequality (9.5) then implies that $\sum_{u \in U(G,G_1)} \sum_{e \in A^-(u)} y^j_e \geq r(G)\lceil \mathrm{OPT}_f(G_1) \rceil$. This establishes that the flow y^j satisfies all the constraints in the LP (CR). Since c^j is a circulation, $|y^j| = |y'|$, and hence y^j is also an optimal solution to the LP (CR).

The construction of the two circulations c^1 and c^2 ensures that for at least one edge e in the fractional graph $G_{y'}$, the variable y^j_e takes on an integer value in the optimal solution y^j to (CR). Thus, the fractional graph G_{y^j} has strictly fewer edges than $G_{y'}$. We can repeat this cycle-canceling process, in each iteration obtaining a different optimal solution to (CR), and decreasing the number of edges in the fractional graph by at least 1. After at most $|E(G)|$ iterations, we will have an optimal solution y to (CR) for which the fractional graph G_y is acyclic.

Proof of Theorem 3.2

For any module G in the containment tree, consider the subtree of the containment tree rooted at G. We claim that the collection of flows $\{f^G\}$ computed by Algorithm TR satisfies the edge coverage and weak coupling properties in all modules in the subtree, and that $|f^G| \leq r(G)\lceil \mathrm{OPT}_f(G) \rceil$. Combined with Proposition 2.1, this implies that the algorithm outputs a complete test set for H of size at most $|f^{M_1}| \leq r(M_1)\lceil \mathrm{OPT}_f(M_1) \rceil \leq r(M_1)\mathrm{OPT}(M_1)$.

The proof of the above claim is by induction on the depth of G. In the base case, $d(G) = 1$, the constraints in the instance of the optimization problem (MC) used to compute the flow f^G ensure that it covers all the edges of G, and $|f^G| = \mathrm{OPT}_f(G)$.

For the inductive case, we have $d(G) > 1$. The children $G_1, \ldots, G_k$ of G in the containment tree have depths smaller than $d(G)$, and thus the induction hypothesis states that for any $i = 1, \ldots, k$, the collection of flows computed by Algorithm TR satisfies the edge coverage and weak coupling properties in the subtree rooted at G_i, and $|f^{G_i}| \leq r(G_i)\lceil \mathrm{OPT}_f(G_i) \rceil$.

We only show the proof here for the case $r(G) = r(G_1)$. This implies that $r(G_1) > r(G_2)$, and so $r(G_i) \leq r(G) - 1$ for all $i = 2, \ldots, k$. The cycle-canceling procedure used by the algorithm results in an optimal solution y to (CR). Algorithm TR then constructs a source-sink flow f^G from y such that $f^G_e \geq \lfloor y_e \rfloor$ for all $e \in E(G)$. By the constraints (3.2) in the LP (CR), we have $\lfloor y_e \rfloor \geq \max\{\lceil (r(G) - 1)x_e \rceil, 1\} \geq 1$, where x is an optimal solution to the LP (CF) for G with value $|x| = \mathrm{OPT}_f(G)$. Hence, $f^G_e \geq 1$, and the edge coverage property is satisfied in the subtree rooted at G.

For all $i = 2, \ldots, k$, the constraints in the LP (CF) and the induction hypothesis imply the weak coupling property for G_i.

$$\begin{aligned}\sum_{u \in U(G,G_i)} \sum_{e \in A^-(u)} f_e^G & \\ &\geq \sum_{u \in U(G,G_i)} \sum_{e \in A^-(u)} (r(G)-1)x_e \\ &\geq r(G_i)\lceil \mathrm{OPT}_f(G_i)\rceil \geq |f^{G_i}|\end{aligned}$$

In the case of G_1, the constraint (3.1) in the LP (CR) ensures that $\sum_{u \in U(G,G_1)} \sum_{e \in A^-(u)} y_e \geq r(G)\lceil \mathrm{OPT}_f(G_1)\rceil$. If the fractional graph G_y has no edges, then f^G is equal to y, and therefore the weak coupling property is satisfied for G_1. Otherwise, G_y consists of a single path P from the source to the sink. Let the vertices on this path be $s(G) = v_1, \ldots, v_q = t(G)$. As in the proof of Lemma 3.3, for any $i = 1, \ldots, q-1$, if the edge (v_i, v_{i+1}) is in $E(G)$, then it is referred to as an ascending edge, and if (v_{i+1}, v_i) is in $E(G)$, then it is called a descending edge.

Let P^A (P^D) be the set of ascending (descending) edges on the path that enter supernodes that map to G_1. Define $\alpha = \lceil y_e \rceil - y_e > 0$ for an ascending edge $e \in P^A$ (as the in-degree of the source is 0, there must be an ascending edge in P^A). The algorithm will increase the flow on ascending edges of P by α, and decrease the flow on descending edges of P by α, to obtain a source-sink flow f^G of value $\lceil |y| \rceil$. If it were the case that $|P^A| < |P^D|$, then y would not be an optimal solution to (CR), because it would be possible to decrease the flow on the ascending edges and increase the flow on the descending edges in P to obtain a new feasible solution to (CR) of value strictly less than $|y|$. Since y is an optimal solution, we must therefore have $|P^A| - |P^D| \geq 0$. From this inequality and the fact that f^G and y are equal for all the edges in G entering supernodes that map to G_1 other than those in P^A and P^D, we obtain the following bound.

$$\begin{aligned}\sum_{u \in U(G,G_1)} \sum_{e \in A^-(u)} (f_e^G - y_e) &= \sum_{e \in P^A \cup P^D} (f_e^G - y_e) \\ = \sum_{e \in P^A} \alpha + \sum_{e \in P^D} (-\alpha) &= \alpha(|P^A| - |P^D|) \geq 0\end{aligned}$$

$$\begin{aligned}\sum_{u \in U(G,G_1)} \sum_{e \in A^-(u)} f_e^G &\geq \sum_{u \in U(G,G_1)} \sum_{e \in A^-(u)} y_e \\ &\geq r(G)\lceil \mathrm{OPT}_f(G_1)\rceil \\ &= r(G_1)\lceil \mathrm{OPT}_f(G_1)\rceil \geq |f^{G_1}|\end{aligned}$$

We conclude that the algorithm computes a collection of flows that satisfies the weak coupling property in the subtree rooted at G. The value of the flow f^G is $|f^G| = \lceil |y| \rceil$, which by Lemma 3.2 is bounded from above by $|f^G| \leq \lceil r(G)|x| \rceil = \lceil r(G)\mathrm{OPT}_f(G)\rceil \leq r(G)\lceil \mathrm{OPT}_f(G)\rceil$.

Proof of Theorem 4.1

We show that the collection of flows $\{f^{M_i}\}$ computed by Algorithm D satisfies the edge coverage and weak coupling properties, from which it follows by Proposition 2.1 that the output of the algorithm is a complete test set for H. Consider any module M_i in the topological order defined by the containment graph, where $i \in \{1, \ldots, K\}$. The algorithm computes an optimal solution (x, z) to the linear program (CD) such that $|x^{M_1}| = \mathrm{OPT}_f$. The lower bounds ℓ_e in the minimum cost circulation instance (MC) used to compute f^{M_i} ensure that for any edge $e \in E(M_i)$, if $d(M_i) = 1$, then $f_e^{M_i} \geq \ell_e = 1$, and if $d(M_i) > 1$, $f_e^{M_i} \geq \ell_e = \lceil (d(M_i)-1)x_e^{M_i} \rceil \geq (d(M_i)-1)x_e^{M_i} \geq x_e^{M_i} \geq 1$. This shows that $\{f^{M_i}\}$ satisfies the edge coverage property.

We now verify that for any module M_j, $j \in \{2, \ldots, K\}$, the amount of flow entering supernodes that map to M_j is at least $|f^{M_j}|$. If $d(M_j) = 1$, then the flow x^{M_j} is feasible for the minimum cost circulation instance (MC) with lower bounds $\ell_e = 1$, and so $|f^{M_j}| \leq |x^{M_j}|$, as f^{M_j} is an optimal solution to this instance. In the case that $d(M_j) > 1$, Lemma 3.1 implies that $|f^{M_j}| \leq d(M_j)|x^{M_j}|$. Thus, the inequality $|f^{M_j}| \leq d(M_j)|x^{M_j}|$ is satisfied in both cases.

The constraints (4.4) in the LP (CD) ensure that $\sum_{i=1}^{j-1} z_{ij} \geq |x^{M_j}|$. Using the equalities (4.3) in the LP, we substitute for the z_{ij} variables to obtain $\sum_{i=1}^{j-1} \sum_{u \in U(M_i,M_j)} \sum_{e \in A^-(u)} x_e^{M_i} \geq |x^{M_j}|$. If a module M_i contains M_j, then we must have $d(M_i) \geq d(M_j)+1$. For every edge $e \in E(M_i)$ for which there is a term $x_e^{M_i}$ in the summation, the lower bounds in the minimum cost circulation instance enforce the inequality $f_e^{M_i} \geq (d(M_i)-1)x_e^{M_i} \geq d(M_j)x_e^{M_i}$.

$$\begin{aligned}|f^{M_j}| &\leq d(M_j)|x^{M_j}| \\ &\leq \sum_{i=1}^{j-1} \sum_{u \in U(M_i,M_j)} \sum_{e \in A^-(u)} d(M_j)x_e^{M_i} \\ &\leq \sum_{i=1}^{j-1} \sum_{u \in U(M_i,M_j)} \sum_{e \in A^-(u)} f_e^{M_i}\end{aligned}$$

We conclude that the collection of flows $\{f^{M_i}\}$ satisfies the weak coupling property, and that Algorithm D produces a complete test set for the hierarchical model H. Noting that the size of the test set is the value of the flow in the top-level module M_1, we bound the size of the test set by $|f^{M_1}| \leq d(M_1)|x^{M_1}| = d(M_1)\mathrm{OPT}_f \leq d(M_1)\mathrm{OPT}$.