A New and Fast Approach to Very Large Scale Integrated Sequential Circuit Test Generation

Author(s): J. B. Adams and D. S. Hochbaum

Source: *Operations Research*, Vol. 45, No. 6 (Nov. - Dec., 1997), pp. 842-856
Published by: INFORMS
Stable URL: http://www.jstor.org/stable/172069
Accessed: 04-11-2017 22:47 UTC

A NEW AND FAST APPROACH TO VERY LARGE SCALE INTEGRATED SEQUENTIAL CIRCUIT TEST GENERATION

J. B. ADAMS and D. S. HOCHBAUM

The University of California, Berkeley, California

(Received January 1995; revisions received July 1995; accepted September 1995)

We present a new approach to automatic test pattern generation for very large scale integrated sequential circuit testing. This approach is more efficient than past test generation methods, since it exploits knowledge of potential circuit defects. Our method motivates a new combinatorial optimization problem, the *Tour Covering Problem*. We develop heuristics to solve this optimization problem, then apply these heuristics as new test generation procedures. An empirical study comparing our heuristics to existing methods demonstrates the superiority of our approach, since our approach decreases the number of input vectors required for the test, translating into a reduction in the time and money required for testing sequential circuits.

Very large scale integrated sequential circuit testing is necessary to ensure the reliability of circuits. Since circuits contain from thousands to millions of parts and connections, testing of sequential circuits is currently a time-consuming and expensive process, which needs to be made more efficient. Even small reductions in time invested for testing lead to savings in the manufacturing process.

At the termination of a typical manufacturing process thousands of chips that contain the physical representation of a circuit need to be tested for functional and logical correctness. For each unit produced the testing should determine if the chip is faulty or not.

Very large scale integrated sequential circuit testing thus attempts to pinpoint problems with a circuit, determining whether faults are present. However, this task becomes difficult when the circuit cannot be examined internally. If the circuit is viewed as a black box to which we can apply input and then observe output, but never see the internal workings, then determining exactly whether the circuit is faulty is challenging.

One way of testing a circuit (or the corresponding chip) is to try out all possible inputs and compare the resulting outputs to the outputs of a fault-free circuit. This process is very time consuming, and yet it works only for *combinational* circuits, those that have no feedback mechanism (see Figure 1(a)). In a *sequential* circuit there is a feedback mechanism which implies that it is not sufficient to try all possible inputs, but also all possible values of the memory elements have to be tried in conjunction with the inputs. Consequently, the testing of sequential circuits is far more complex than the process of testing combinational circuits, which is in itself a difficult problem.

In an attempt to reduce the complexity of sequential circuit testing, internal testing methods have been devised to "open up" the black box. Techniques such as Scan Path, or Level Sensitive Scan Design, which employ shift register latches to control and observe all internal states (Williams and Parker 1983) can be used. Since these methods are costly, time-consuming, and potentially damaging to the circuit, they are avoided unless absolutely necessary.

Our goal here is to propose a more efficient method for testing circuits that does not monitor the internal state at every step, yet still exploits knowledge of the circuit interior in order to come up with a faster test. Thus, our approach is useful for testing circuits for which the physical design is known. The information about the circuit is represented as a list of potential defects and the corresponding resulting performance errors.

In the evaluation of testing methods, the amount of time required to create the test vectors is of secondary importance since the creation of the test vectors is done once for each circuit design. The efficiency is measured in terms of the number of generated test vectors, as this number is proportional to the amount of time required to test a given circuit chip. The manufactured circuits are tested with the same set of test vectors repeatedly (and commonly thousands or even millions of times), and therefore the number of test vectors is of prime importance.

Testing of circuits can be viewed as comprising two main tasks. One is the generation of inputs that can lead to external manifestation of existence of faults; the other is the compression, or compaction, of this set of vectors. These two tasks are usually viewed as one, and the attempt is to generate a compacted list of test vectors. Our approach allows for the separation of these tasks. It can work with any test generation procedure and produce a faster test than would have been produced by the test generation alone.

The approach we use is proved faster than commonly used approaches on the set of benchmark circuits ISCAS-89. These circuits are considered the standard benchmark

Subject classifications: Industry, computer electronic: semiconductor manufacturing. Manufacturing performance/productivity: improved quality control by effective testing. Programming integer heuristic: heuristic approach for the tour cover problem.
Area of review: MANUFACTURING, OPERATIONS AND SCHEDULING.

0030-364X/97/4506-0842 $05.00

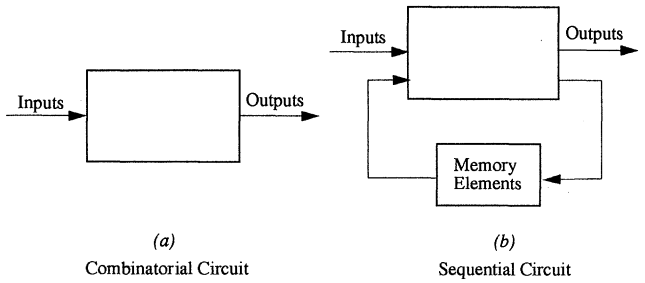

Figure 1. Schematic view of combinatorial circuit versus that of sequential circuit.

for algorithms on sequential circuits. Real circuits are substantially larger than the benchmark circuits, yet the methods used are the same as for the small ones, typically combined with a heuristic decomposition of the large circuit to a collection of smaller ones. The method we propose can similarly be used in conjunction with such decomposition and is likely to be faster on the very large real circuits as well.

The testing method we develop is based on modelling the testing problem as a tour covering problem. In this problem the input is a directed graph and a set of elements. Each arc has an associated subset of elements it covers. The problem is to find the shortest length tour that traverses arcs which jointly cover all the elements. As we point out, this model has some deficiency and does not represent fully all aspects of the testing problem. We correct for those deficiencies in the heuristic procedures devised for solving the problem in the testing context.

PRELIMINARIES

Figure 1(b) presents a schematic description of a sequential circuit. This type of circuit has a feedback mechanism. Logic values in the next-state lines are stored in memory units and then fed back into the circuit as the present-state values in the next timeframe. These memory units are also referred to as latches or flip-flops (denoted by FF in Figure 2). Figure 2 presents one time frame of a sequential circuit. The values carried by the latches along with those assigned to the primary inputs determine uniquely the outputs *and* the next state of the circuit.

The gates of the circuit in Figure 2 represent the well-known Boolean functions, AND, OR, and NOT. For two (or more, if the gates are concatenated) 0 − 1 inputs the AND function outputs 1 if and only if all inputs are 1. The OR function outputs 1 if and only if at least one input is 1. The NOT function has a single input, and it reverses 1 to 0 and 0 to 1.

The model best suited for describing the behavior of a VLSI sequential circuit is the *finite state machine*, a five-tuple, $M = (S, \mathcal{I}, \mathcal{O}, v, w)$ (Grimaldi 1989). S is the set of the states of the machine, $\mathcal{I}$ is the input alphabet, $\mathcal{O}$ is the output alphabet, v is the next-state function, $v{:}S \times \mathcal{I} \rightarrow S$, and w is the output function, $w{:}S \times \mathcal{I} \rightarrow \mathcal{O}$. Logic values

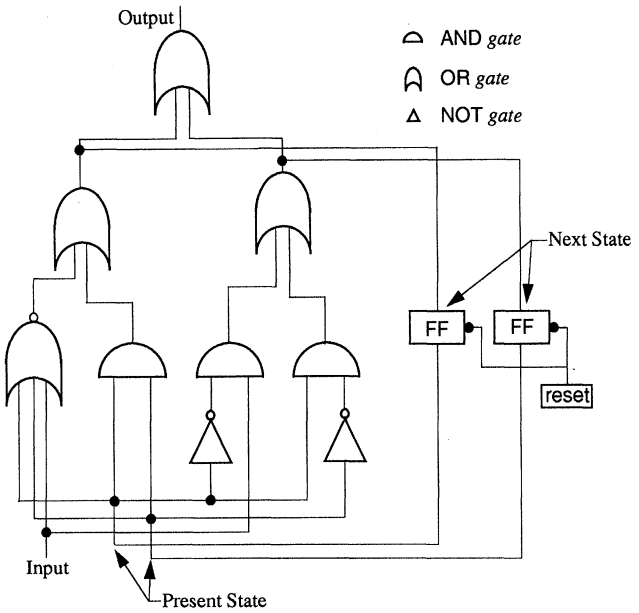

Figure 2. Sequential circuit.

in the circuit's present-state lines correspond to the machine's current state, input applied to the circuit is identical to the machine's input.

The finite state machine associated with a circuit defines its *State Transition Graph* (STG) which conveniently and concisely encodes information about all possible states, their allowable inputs, and reachable next-states (Mano 1991 and Ghosh et al. 1991). Nodes in the graph correspond to logic states in the circuit, while arcs represent transitions between states. Thus, each node has outgoing arcs representing transitions resulting when allowable input is applied to that state. Arcs are labeled with this input, as well as the output produced when this input is applied. Movement from one state to another in the State Transition Graph simultaneously relates to transitions of the finite state machine and to the functioning of the circuit. We will refer occasionally to states as nodes and to transitions as arcs.

Each input vector applied to a circuit at a given state produces an output and a transition to a different state. While the output is externally visible, the resulting state is invisible. This invisibility of the state plays an essential role in all testing methods for sequential circuits.

For further clarification we digress at this point to explain the notion of *don't cares*. Suppose the inputs include $q_1, \ldots, q_n$ as the primary inputs, and $q_{n+1}, \ldots, q_m$ as the memory latches values (the state). Suppose q_1, q_2 are inputs to an OR gate and $q_1 = 1$. Then no matter whether q_2 is 0 or 1, the outgoing line from the gate will carry the value 1. In this case we say that q_2 assumes a "don't care" value, which is denoted by X. Similarly if q_{n+1}, q_{n+3} are inputs to an AND gate and $q_{n+3} = 0$, then the output line from the gate is always 0 and $q_{n+1} = X$. By the same

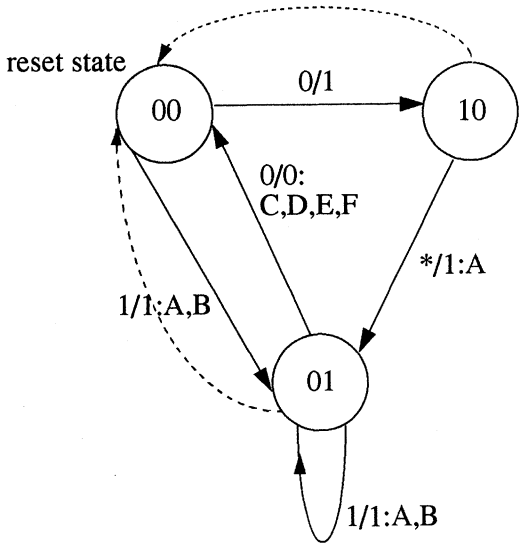

Figure 3. State transition graph (STG).

token, many of the state values can be don't-care values. Consequently each state can be viewed as one of an equivalence class of states that have the same pattern of transitions given the same inputs (or the same equivalence class of inputs). This view of each state as a set and each input vector as a set, exploiting the existence of don't cares, is significant in reducing the complexity of dealing with large circuits. We will refer to each state and input vector as a singleton, whereas actually each may represent a set of states or input vectors.

While we do not make explicit use of the notion of don't cares, this is an important aspect of the subroutines we use for test vector generation per faults and for fault simulation.

The State Transition Graph contains one or more *reset states*, states whose logic values are equal to values in the circuit's present state when it is first switched on. An STG has more than one reset state if its equivalence class includes more than the state itself, or alternatively, if some of the present state bits are "don't cares" (Mano 1991). Any state in the STG can be reached from reset. Otherwise, if a state cannot be reached it is not a valid state and is not included in the STG. Further, the circuit can be switched off and then on again no matter what values are in the present state lines. Thus reset is reachable from any state, making the STG a strongly connected digraph. It should be noted, however, that a cost—possibly greater than that of making one finite state machine transition—is associated with restarting the machine to reach the reset state (Pixley et al. 1992).

Figure 3 depicts the STG associated with the circuit of Figure 2. Outgoing arcs in Figure 3 are labeled with possible input and the corresponding output, as well as the index letters of defects whose presence alters output along these arcs. For instance, if arc (j, k) is labeled with $i/o : l_1, l_2, \ldots, l_L$, then a transition from state j to state k occurs when the machine is fed input i. Output o is produced, and the transition tests if faults $l_1, l_2, \ldots, l_L$ are present. Dashed arcs leading out of every state to reset represent making a transition to the reset state by switching the machine off then on again.

Faults in the circuit can affect its behavior, either by altering the output produced in a state or by affecting transitions out of the state, sending the circuit to the incorrect next-state. With the *single stuck-at* model, the fault model we work with, a fault is present when a circuit wire's logic value is corrupted so that it is fixed either at value zero or one. It is further assumed that only one such fault is ever present at a time in a circuit (Williams and Parker 1983). This standard assumption may lead to invalid tests as is manifested in the veering off phenomenon that we address in detail in Section 4.

An *excitation vector* for a fault is an assignment of values to the circuit's present state lines and primary inputs, which produces faulty output or causes an incorrect next-state transition when the fault is present. An *excitation state* is the present state part of the excitation vector. A sequence of input vectors that takes the machine from the reset state to an excitation state, while the defect present is known as a *justification sequence*, and the process of sending the machine to the excitation state is referred to as *justifying* the state (Ghosh et al. 1991). A *test* sequence for a fault is a sequence of input vectors, which when applied to the faulty circuit, produces output values different from those the fault-free machine produces. If a fault alters output produced at an excitation state, the justification sequence concatenated with the input part of the excitation vector form a test for the fault. However if a fault alters the excitation state's next-state but not its output, the justification sequence alone does not form a test for that fault, since the effects of the defect cannot be observed at the excitation state. Additional input vectors must be applied, until the effects of the fault are propagated to the output lines. Such a sequence of input vectors applied at the excitation state which propagate the effects of the fault to the output are termed a *differentiating sequence* (Ghosh et al. 1991). Concatenated to the justification sequence and input part of the excitation vector, these form a test.

We refer to the sequence beginning with the excitation state and ending with the state ending the differentiating sequence as ED-path. The sequence beginning at reset followed by the justification sequence and the ED-path we call JED-path. In order to clarify why the ED-path may consist of more than one arc (transition) consider the example in Figure 4. That example depicts a JED-path, which is a test for fault f, as followed in a fault-free circuit versus the same path followed in the presence of fault f. At the point where the excitation state is reached the behaviour of the circuit differs in the faulty circuit from that in the fault free circuit. Figure 4 shows the two paths that are followed in both circuits. The faulty one follows the dotted transitions while producing outputs O_j that are

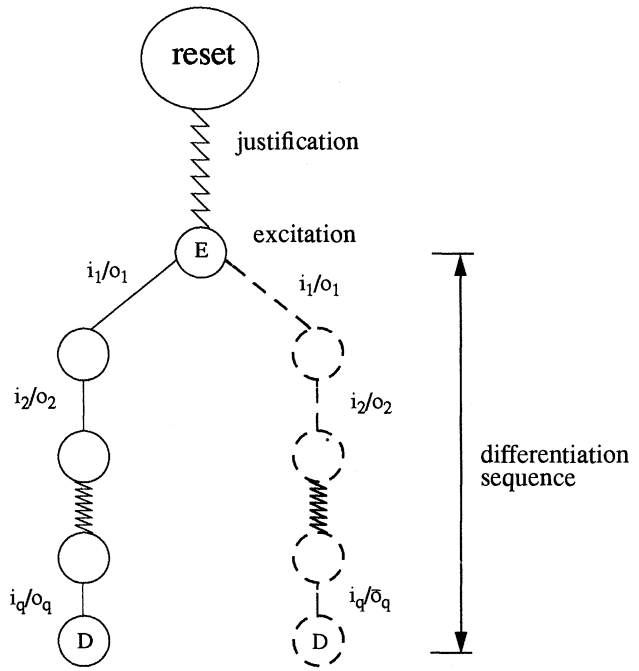

Figure 4. The JED path followed in both correct and faulty circuits.

identical to the outputs produces in the fault-free circuit until the differentiating transition leading to D. On this final arc of the path the output of the fault-free circuit is O_q whereas the output of the faulty circuit is $\bar{O}_q$. So although the fault is present, it is not manifested prior to reaching state D.

The task set by the testing problem is to traverse the state graph with a minimum number of input vectors so that for each potential defect one of its excitation states is visited, and immediately thereafter its corresponding differentiating sequence is applied. For the circuit of Figure 2, one possible test checking for defects A through F starts at reset, or state 00, and consists of the input vector sequence 1–0.

There is more than one dimension to determine the quality of a test procedure. It ought to be fast and require on the average a small number of input vectors per fault checked. Another factor determining the quality of testing is that a large fraction of faults (ideally all) are detected. It has been observed empirically that any crude approach to testing will be able to test a small fraction of the faults very quickly (with small number of test vectors). As more and more faults are checked, the number of input vectors needed to detect the next fault tends to go up. Our goal is therefore to devise a test procedure that is both fast *and* detects a large proportion of the set of faults compared to other test procedures.

LITERATURE REVIEW

Different approaches have been taken for conducting VLSI sequential circuit test generation without prying into the internal workings of the circuit, and thus by justifying states and checking output. One of the most popular methods, due to its simplicity, is the *random approach*. The random approach is to generate sequences of input vectors composed of zeros and ones at random. These input vectors are applied to faulty circuits while the output is observed. If this output differs from that which a fault-free circuit would produce, as determined from the STG, the vector forms a test for the defect in the circuit. Random tests do manage to check for some defects. However, they are not efficient, at least not after a small fraction of faults have been detected. (Our observation is that this fraction is about 20 percent.) It is possible, using information about the circuit structure, to design smarter tests which check for more faults using less input vectors.

Ghosh et al. (1991) present such a test procedure. Their method finds several paths, each starting at the reset state and traveling through the STG, checking for a fault. The approach justifies and differentiates states one at a time, starting at the same initial reset state before each justification sequence. Because of this strategy of looking for input vectors that check for faults one at a time, we refer to their method as the *single-fault approach*. The idea of repeatedly resetting the circuit (or the machine) to this initial state before looking for a sequence for each defect proves wasteful, however. In addition to adding distance to the test vector, resetting every time adds cost to the testing process since the machine must be switched off then on again to get back to the initial state. As we show later, it is frequently beneficial to not to restart and look for an entirely new test sequence for each fault. Instead, simply adding a few input vectors to an existing sequence could take the machine a few states further and check additional faults. Such an approach would be less costly and time-consuming than generating a new test sequence for each defect, not using information from previous defects. Ghosh et al. have demonstrated, however (and our empirical studies have confirmed), that their approach is substantially more effective than the random approach.

Füredi and Kurshan (1987) proposed a theoretical method that views test generation as a requirement to traverse every arc in the graph. As such, they solve the Chinese Postman Problem, or the problem of finding a minimum length tour that traverses every arc in the graph, defined on the STG graph. (We present a formal definition of the Chinese Postman Problem in Section 2.) We refer to such an approach as a *Chinese Postman approach*. This strategy has several shortcomings. By requiring that every arc in the STG is traversed, this approach traverses more than one edge for each defect and also traverses edges that do not differentiate any defect at all. The method is also prone to error due to the phenomenon of *veering off*. This happens when one gets the right output, but it is produced at the wrong state. More explicitly, the test procedure is based on the logic description of the fault-free circuit and depends on the machine being in the correct state at every

step. Since faults may cause incorrect next-state transitions, the machine may not always move into a planned state, but to a different state, due to a faulty transition. Then it has veered off and it may not be possible to justify all excitation states at once. Although the circuit may appear, deceptively, to produce the correct output vector, this happens only because it has veered off, and the output at this new state just happens to coincide with the correct output at the planned state.

Another shortcoming of the method of Füredi and Kurshan (1987) is that it requires the entire STG to be generated in advance. Even for small size circuits, with only 10–15 memory latches, this is impossible. For these reasons this approach is not implementable as is. The concept, however, is useful and we incorporate it in our approach.

Aho et al. (1988) employ an improvement on the Chinese Postman approach. They seek to cover only edges that cover defects, i.e., those that correspond to differentiating vectors. The problem of covering a subset of edges with a single tour of minimum length is called the Rural Postman Problem, and we call this approach to testing the *Rural Postman* approach. The application of this problem discussed in Aho et al. comes up in a related but different context. They generate tests for checking the conformance of a protocol implementation to its specification, a problem closely related to sequential circuit testing. Their problem, however, requires that all of a certain subset of transitions of a Finite State Machine be checked, while our problem must check only selected transitions, since a fault can have more than one ED-path. For the testing problem, their approach has the shortcoming of traversing a differentiating sequence for a fault even after that fault may have already been identified. Again, this approach is not practically implementable for the same reasons as the previous one—the veering off and the explicit generation of the STG.

There are also more general test generation methods, appropriate when a list of potential defects and differentiating vectors is not available. Such is the case when testing implementation of logical design, or testing protocols. Lee and Yannakakis (1992) present such a testing method taking as input two machines, A and B, the specification machine, and the implementation machine, respectively. They construct an input sequence that distinguishes the two machines. This sequence is of length $O(pn^4 \log n)$, where n is the number of states in the STG and p is the number of input lines. That is, if machine B contains some defect, their test will find it in time proportional to more than a fourth-degree polynomial of the size of the state space. Because Lee and Yannakakis's method works with machines about which little is known, their method is not constraining and thus has wide-ranging applications. It does, however, require the traversal of all edges in the graph (at least once), as it does not assume the availability of information about the circuit and its potential defects. It is therefore prohibitively slow and inappropriate for checking physical circuits.

Among these existing methods only the random and single-fault methods are practical for circuits of moderate size. We will therefore compare our approach to these two methods.

OVERVIEW

Our approach seeks to traverse at least one differentiating sequence for each defect, so that the number of input vectors applied (and hence the number of edges traversed) is as small as possible. To achieve this end we examine combinatorial aspects of the test generation problem.

We begin by presenting in the next section the relevance of the set covering problem to the testing problem. A discussion on the similar features of the VLSI sequential circuit testing to the crew scheduling problem implies that similar empirical approaches may be used for solving both problems. In the following section we introduce a new combinatorial optimization problem, the *Tour Covering* problem, in which aspects of the Rural Postman are combined with set covering aspects of the testing problem. Three heuristics for Tour Covering are introduced in the third section, including a greedy heuristic that generates test sequences online. Section 4, *Implementation for Sequential Circuit Testing*, relates this optimization problem back to sequential circuit testing, compensating for the veering off phenomenon. Section 5 presents results from an empirical study that applies three new testing heuristics to benchmark circuits. Data from these three methods are compared with data from the single-fault approach and the random approach. Finally, the last section presents conclusions from the study and plans for future research.

1. THE SET COVERING APPROACH

The set covering problem can be viewed as a simplified version of the testing problem. Consider the path created in the STG graph by traversing from the reset state the justification sequence/excitation state/differentiating sequence for a given fault. Recall that such path is termed JED-path to distinguish it from a path starting with the excitation state/differentiating sequence, which we call an ED-path. A path usually detects a collection of faults. For a path $\mathcal{P}_j$ (of either type) we denote the collection of faults detected by the path as $F(\mathcal{P}_j) \subseteq F$, where F is the set of all faults. We will consider the cost of a path to be the number of vectors in the path, or in other words, the length of the path $|\mathcal{P}_j|$. A path can therefore be viewed as a set that covers a subset of the faults. With this view in mind we can formulate the problem as a set covering problem using the notation

$$a_{ij} = \begin{cases} 1 & \text{if fault } i \in F(\mathcal{P}_j), \\ 0 & \text{otherwise,} \end{cases}$$

and the variable x_j is set to 1 if path $\mathcal{P}_j$ is selected for the testing and 0 otherwise.

The set covering problem is then formulated as

$$\min \sum_{j=1}^{m} |\mathcal{P}_j| x_j$$

(SC)

$$\text{s.t.} \sum_{j=1}^{m} a_{ij} x_j \geq 1,\ i \in F,\ x_j \in \{0, 1\}, \quad j = 1, \ldots, m.$$

Consider now the case when the paths are JED-paths, and hence their length is calculated from reset. In that case the optimal solution to (SC) is an upper bound on the length of the test sequence. This is because it is possible to concatenate another ED path to detect more faults, but this formulation does not allow for that. In that sense, this formulation emulates an optimization version of the single-fault method. After detecting some faults and reaching a differentiating state, it returns to reset.

The case of using ED-paths is more intricate. The (SC) formulation then ignores the additional cost required to get from the end of one path (the differentiating state) to the beginning of another (the excitation state). In that sense the optimal solution provides a lower bound. On the other hand, various ED-paths may be overlapping, and hence their combined length is less than actually stated. For these reasons the value of (SC) when we consider ED-paths is neither a lower nor an upper bound.

One way to remedy that is to check for all possible overlaps. This in itself may be too time consuming to be practical. However, one can heuristically just check for overlapping pairs and if any are found, add the union of the two paths to the set of candidate paths. Assuming that the number of overlaps is not substantial, this can be a good "approximate" lower bound.

Checking, to a limited extent, for overlaps is useful also in the case we use JED-paths. There we can check for the overlap of ED-paths and add the shortest justification sequence to the concatenation of the overlapping paths. This will result in an improved (i.e., lower in value) upper bound. The ultimate improvement would be to check for all possible successive routings of ED-paths and use their union as additional paths for the (SC) model. This, however, is too time consuming to be practical.

An additional aspect that is ignored in the set covering model is the veering off effect. In addition to these shortcomings, the set covering is a well known NP-complete problem. As such, even this simplified version of the testing problem is intractable.

The set covering instances that we consider have some specific features. The number of sets is typically extremely large, as there are many different paths that could potentially test the same defect. In fact, it is practically infeasible to enumerate all such potential paths. A set covering problem with the same features comes up in the context of the crew scheduling problem for airlines. There, the sets are all the possible individual crew schedules, each specified in terms of the flight legs that it includes. The number of such schedules is typically exponential. In practical solution methods developed for the crew scheduling problem, the set of potential schedules is generated as needed (the column generation technique), rather than being given with the initial input. Such an approach is most appropriate for our application as well, in which context it means that test vectors are not specified in advance.

There exist some good recent methods for solving the crew scheduling problem. In particular, we believe that a column generation technique, such as the one developed for the crew scheduling problem by Anbil et al. (1991) could result in a small collection of test sequences that forms a comprehensive test for the testing problem. We do not pursue this direction here, but consider it very promising for future research.

2. THE TOUR COVERING PROBLEM

The tour covering problem introduced in this section is a better model of the testing problem compared to the set covering. It is still a simplified model of the problem, as will be explained later. The tour covering problem combines the set covering aspects of the testing problem with the requirement that edges must be traversed along a contiguous path that starts at the reset state, or along a tour if the path also terminates at reset.

A relevant problem here is the Chinese Postman Problem (CPP), that was initially solved by Edmonds and Johnson (1973). In CPP, we are given a directed graph (the undirected version is irrelevant for our discussion), with each arc having some weight, representing its length, associated with it. The goal is to create a tour that traverses all the arcs so that the total length of the tour is minimum. This problem is solvable in polynomial time by reducing it to a transportation problem, as discussed in the next section.

A more closely related problem is the Rural Postman Problem (RPP). Here there is a specified *subset* of the arcs in the graph that need to be traversed in a tour. The other arcs may or may not be included in the tour. Although a minor variation of the CPP algorithm finds, in polynomial time, a collection of tours of minimum length traversing all edges in the subset at minimum length, the problem of creating a *single* tour of minimum length is NP-complete (Aho et al. 1988). (Indeed, it is easily seen that the Traveling Salesperson Problem is reducible to it by setting each subtour as a node in a graph.)

As in RPP, in the testing application we need to traverse only a subset of the edges, namely those that detect faults. Moreover, we need to traverse only one of the set of many paths that can detect a certain fault. This is the problem we call the *Tour Covering* problem, and it is defined formally as follows:

Tour Covering Problem

Instance: A directed graph, $G = (N, A)$, a universal set F and subsets of F associated with each arc $a_j \in A$, $F(a_j)$.

Problem: Find a shortest length (not necessarily simple) cycle which starts at a prespecified node and travels through the graph, traversing a subset of the arcs $\bar{A} \subseteq A$ such that $\cup_{a \in \bar{A}} F(a) = F$.

The Tour Covering problem is related to set covering, since we seek to cover all of the n elements. Also it generalizes the Rural Postman Problem, which requires finding a shortest tour traversing a certain subset of the arcs.

The Tour Covering problem when used in the context of testing has faults detected associated with arcs. In fact the detection of faults requires traversing a sequence of arcs in an ED-path—from excitation to differentiation. Another aspect that is not addressed in the Tour Covering formulation is the veering off effect. In an ideal testing procedure the length of the tours should be bounded to reduce the likelihood of veering off. In that sense a collection of a small number of tours, each beginning and ending at reset, is preferred to a single tour. This issue will be discussed in detail in the implementation section (Section 4). The additional number of vectors required for the multitour covering still justifies the added certainty that veering off is avoided.

3. HEURISTIC ALGORITHMS FOR THE TOUR COVERING PROBLEM

3.1. The Chinese Postman Approach

Suppose one applies a Rural Postman algorithm to solve the tour covering problem as follows. Consider each arc a that has a nonempty set $F(a)$ associated with it to require traversing. The union of these arcs is then the subset of arcs that need to be traversed. In that case the solution tour covers all these arcs, although many may not need traversal as the set they cover has already been covered by other arcs. To avoid that, we generate in our heuristic one subtour at a time, and then update the remaining set of elements to be covered on the remaining arcs in the graph.

When the CPP algorithm is applied to the Rural Postman Problem, the result is in general a collection of subtours of total minimum weight among all possible such collections that cover all edges in the specified subset. Although such a result is inappropriate as a solution to the Rural Postman Problem, it is better suited for the purposes of the Tour Covering problem than a single tour. In our heuristic we will use the CPP algorithm and then select among all the subtours the one subtour that contributes most to the covering, where the contribution is measured in terms of the number of *new* elements covered per unit weight of the subtour. At each iteration a subtour is selected in this way. Then the collection of elements to be covered is updated. Those covered by the selected subtour are removed from the collection. When terminating the heuristic returns the collection of selected subtours, which together cover all n elements.

The heuristic calls for a routine CPP which generates a collection of tours covering the arcs in the subset at minimum total cost. That algorithm's description is given later in this section.

CPP Heuristic for Tour Covering

Input: $G = (N, A)$; F, $F(a) \subseteq F$, $\forall a \in A$. $\bar{A} = \{a \in A | F(a) \neq \emptyset\}$. An initial state $v_0 \in N$.

Step 0: (Initialization) $i = 0$, $T = \emptyset$.

Step 1: $i \leftarrow i + 1$.

If $F = \emptyset$ return(T) the set of subtours, STOP.

else {

Call $CPP(G; \bar{A})$

$S_1, \ldots, S_K$ are the subtours returned by $CPP(G; \bar{A})$.

}

Step 2:

For $k = 1, \ldots, K$ {

$C_k = \cup_{a \in S_k} F(a)$

$len(S_k) \leftarrow$ *length of subtour* S_k

}

$|C_{\hat{k}}|/len(S_{\hat{k}}) = \max_{k=1,\ldots,K} |C_k|/len(S_k)$;

If $v_0 \notin S_{\hat{k}}$ *then* $\hat{k} \leftarrow (shortest\_path(v_0, S_{\hat{k}}), S_{\hat{k}}, v_0)$.

$T \leftarrow T \cup S_{\hat{k}}$, $F \leftarrow F \backslash C_{\hat{k}}$.

$\forall \bar{a} \in \bar{A}$, $F(\bar{a}) \leftarrow F(\bar{a}) \backslash C_{\hat{k}}$. If $F(\bar{a}) = \emptyset$ then $\bar{A} \leftarrow \bar{A} \backslash \{\bar{a}\}$.

Go to Step 1.

End

Each subtour generated covers at least one element, as otherwise it could have been omitted while reducing the cost of the solution, thus contradicting the optimality. Hence, after at most n iterations the algorithm terminates.

For the testing problem we place in $\bar{A}$ all the arcs (transitions) that belong to ED-paths generated for all faults. It is then possible that a subtour generated will not include the full ED-path and hence will not test for the faults that this ED-path detects for. For this reason the set of faults that is actually detected by a tour is not simply the union C_k on each subtour as in Step 1 of the algorithm. Instead, we "fault simulated" to identify the actual set of faults detected by each subtour. Details about fault simulation are given in Section 4. Note that it is also possible that more faults can be detected by a subtour compared to the union of sets of faults detected by each arc (or path) in the tour. These will be identified as well by fault simulation.

For completeness, in the remainder of this section we present our implementation of the CPP algorithm following Lawler (1976) used to obtain the collection of subtours. Given a specified subset of arcs to be traversed, the algorithm first determines which additional arcs to include in the subtours. It then constructs the subtours on these edges.

On the graph $G = (N, A)$ let $\bar{A} \subseteq A$ be the specified subset of arcs to traverse. Let N_0 be the set of nodes

adjacent to any arc in $\bar{A}$. A theorem of Edmonds and Johnson (1973) states that a CPP tour, or an Eulerian tour, exists in a strongly connected directed graph the indegree of each node must equal its outdegree. For each node v in N_0, define the indegree, $d_i(v)$ (outdegree, $d_o(v)$) of node v, with respect to arcs in $\bar{A}$, as the number of arcs in $\bar{A}$ whose head (tail) meets that node, respectively. Also define the imbalance of node v with respect to $\bar{A}$, $e(v)$ by the difference between its indegree and its outdegree: $e(v) = d_i(v) - d_o(v)$. If this imbalance is zero, a node is called *balanced.*

Consider N_{pos} and N_{neg}, the sets of nodes in N_0 with positive and negative imbalances, respectively. An analogous version of the theorem of Edmonds and Johnson, appropriate for Rural Postman tours, states that a Rural Postman tour exists if and only if every node is balanced with respect to arcs in these tours. Thus, if some nodes are imbalanced with respect to $\bar{A}$ then the arcs in $\bar{A}$ alone are not sufficient to form a Rural Postman tour.

We consider next the problem of which additional edges to include in the tour. We form a collection A_{pot} of potential additional arcs to be considered for the tour. Then a transportation problem is used to determine which arcs from this collection are actually included in the tour.

Initially A_{pot} is empty. Then we add to A_{pot} the shortest path from v_i to v_j, for each pair (v_i, v_j) of imbalanced nodes with $e(v_i) > 0$ and $e(v_j) < 0$. Note imbalances with respect to A_{pot} of all intermediate nodes on a path are zero, since a path adds exactly one incoming and one outgoing arc to each. Thus including any of the paths in the tour would not disturb the imbalances of these nodes with respect to the tour. To determine which of the paths to include, we solve a transportation problem on the bipartite graph $G_B = (N_{pos}, N_{neg}; A_b)$ which has bipartition N_{pos}, N_{neg} and arc set A_B, consisting of arc (v_i, v_j) for each pair (i, j) such that $e(v_i) > 0$ and $e(v_j) < 0$. Arc (v_i, v_j) has cost equal to the length of the shortest path from v_i to v_j to in G. Node v_i has supply equal to $e(v_i)$.

Let A_{RP} be the final set of arcs to include in the Rural Postman tour. This set consists of arcs in $\bar{A}$ as well as edges in shortest paths whose arcs have positive flow in the transportation problem's solution. The number of copies of arcs on the shortest path between node v_i in N_1 and node v_j in N_2 that are contained in A_{RP} is equal to the value of the flow on arc (v_i, v_j).

Since nodes in N are balanced with respect to A_{RP}, subtours covering arcs in A_{RP} exist. They can be formed by starting at any node in N_0 which is the tail node of a arc and following the arcs along the path of that arc. Once an arc is included in the tour, one of its copies is removed from the set A_{RP}. As long as all of the edges in a arc have a positive number of copies remaining in A_{RP}, the arc is active. When the head node of the arc is reached, arcs along any active arc are followed, one copy of each is removed from A_{RP}, and the process continues until a node is reached with no adjacent active arcs. At this point, a subtour is complete and is placed in the collection of subtours T. Any node in N_0 which is the tail node of an active arc is selected, and the process continues, forming another subtour. Subtours are constructed until no active arcs remain.

Formally, the CPP algorithm is given as follows. It calls a subroutine *shortest_path* which takes two states, v_i and v_j as input, then uses a breadth-first search to find the shortest path between these two states. It also calls *transportation*, which solves a transportation problem on the given graph and returns **X**, an $(N_{\text{pos}}\ N_{\text{neg}})$-dimensional vector of flows, and *eulerian_tour*, which finds and returns an Eulerian tour on the given graph. The output is T the collection of subtours that traverse and cover all arcs in A_{RP}.

CPP Algorithm

Input: $G = (N, A)$ and $\bar{A} \subset A$.

Step 0: (Initialization)

$N_0 = \{v_i \in N \mid (v_i, v_j) \in \bar{A}$, *For some* $v_j \in N\}$.

$e(v) = d_i(v) - d_o(v)\ \forall v \in N_0$,
with $d_i(v)$ and $d_o(v)$ *w.r.t.* $(N, \bar{A})$

$N_{\text{pos}} = \{v \in N_0 \mid e(v) > 0\}$

$N_{\text{neg}} = \{v \in N_0 \mid e(v) < 0\}$

$G_{RP} = (N, \bar{A})$.

Step 1:

For $(v_i \in N_{\text{pos}})$ {
  for $(v_j \in N_{\text{neg}})$ {
    $P(v_i, v_j) \leftarrow$ *shortest_path*(v_i, v_j)
    $l(v_i, v_j) \leftarrow$ *length*$(P(v_i, v_j))$.
  }
}

Step 2:

X $\leftarrow$ *transportation*(G_B), where $G_B = (N_1, N_2, A_B)$, and $A_B = \{(v_i, v_j) \mid v_i \in N_1$ *and* $v_j \in N_2\}$

For $(v_i, v_j) \in A_B$, add x_{v_i,v_j} copies of $P(v_i, v_j)$ to G_{RP}.

Step 3:

$T \leftarrow$ *eulerian_tour* (G_{RP})

return(T).

End

3.2. Nearest Neighbor Approach

The second heuristic for solving the Tour Covering problem is the Nearest Neighbor heuristic. The input is, as before, the set F of n elements to be covered and the subset of arcs $\bar{A}$, each of which covers a nonempty subset of elements.

The nearest neighbor heuristic creates a subtour by appending to a given path starting at an initial node (initial

state or reset) the nearest arc in $\bar{A}$. Finding the nearest arc is determined by calculating the shortest path from the tail of the current path to the head of each arc in $\bar{A}$. The shortest among those shortest paths is appended to the path along with the selected arc. The process is then repeated after updating the set of elements remaining to be covered. If the path folds on itself, it has created a subtour. The process continues nevertheless until all elements of F are covered.

In the algorithm description there is a function $covered\_elements(B) = \cup_{a \in B} F(a)$. For the testing problem this function is evaluated using fault simulation.

Nearest Neighbor Heuristic for Tour Covering

Input: $G = (N, A)$; F, $v_0 \in A$, $\bar{A} \subseteq A$; $F(a) \subseteq F$, $\forall a \in \bar{A}$.

Initialization:

$current\_state$ v_0.

While $(F \neq \emptyset)$ {

$P(a) = shortest\_path(current\_state, starting\_state(a))$.

$P(a_{\min}) = \min_{a \in \bar{A}} P(a)$

$T_{temp} \leftarrow (P(a_{\min}), a_{\min})$, $T \leftarrow (T, T_{\text{temp}})$.

$current\_state \leftarrow end\_state(a_{\min})$

$\mathcal{F}_{cov} \leftarrow covered\_elements(T_{\text{temp}})$

$F \leftarrow F \backslash \mathcal{F}_{cov}$

$\forall a \in \bar{A}$, $F(a) \leftarrow F(a) \backslash \mathcal{F}_{cov}$. If $F(a) = \emptyset$ then $\bar{A} \leftarrow \bar{A} \backslash \{a\}$.

$\bar{A} \leftarrow \bar{A} \backslash T_{\text{temp}}$

}

Return (T)

End

When implementing the nearest neighbor for the testing problem, the arcs in $\bar{A}$ get replaced by the ED-paths with their starting state—the excitation state—and the end state where the fault has been differentiated. We use in this heuristic the terminology of states instead that of head and tail of arcs and paths.

3.3. The Greedy Approach

An approach to Tour Covering that is *on-line* has the advantage that it is not necessary to enumerate all possible paths covering an element ahead of time. Instead, elements can be considered one at a time, and a path to cover the element can be generated as needed. The Greedy Heuristic uses an on-line approach.

This heuristic maintains a set F of elements to cover. In the ith iteration a subtour is formed as follows. The heuristic generates an arc (a ED-path) to cover the first remaining element in F. It then appends the shortest path from the current state to the starting state of the generated arc and then that arc to the tour. At this point the heuristic checks which other elements are covered by the tour or some part of the tour. It removes these elements and the first element from F. Then it generates a path to cover the next element in the updated F, adds the shortest path between the last point in the tour and the first state in the generated path to the tour, then adds the generated path to the tour. It checks which additional elements are covered by the new part added to the tour and removes these and the first one from F. The heuristic continues in this way, until N_i arcs (paths) have been generated and appended to the tour created in the ith iteration. Then the shortest path from the last state in the tour to the specified initial state is added to the tour, completing it. The purpose of closing the tour is to return to the reset state and thus avoid the increasing likelihood of the veering-off effect. Elements which this final path covers are removed from F, completing the ith iteration.

Greedy Heuristic for Tour Covering

Input: $G = (N, A)$; $v_0 \in N$, F, N_i $i = 1, 2 \ldots$.

Initialization: $iter = 0$;

While $(F \neq \emptyset)$ {

$iter \leftarrow iter + 1$

$current\_state \leftarrow v_0$

$T_{\text{iter}} = \emptyset$.

Until $|T_{\text{iter}}| \geq N_{\text{iter}}$,

For $f \in F$ find an arc $\bar{a} \in \bar{A}$ so that $f \in F(a)$.

$T_{\text{temp}} \leftarrow shortest\_path(current\_state, starting\_state(\bar{a}))$

$T_{\text{temp}} \leftarrow (T_{\text{temp}}, \bar{a})$

$current\_state \leftarrow end\_state\ (\bar{a})$

$\mathcal{F}_{\text{cov}} \leftarrow covered\_elements(T_{\text{temp}})$

$F \leftarrow F \backslash \mathcal{F}_{cov}$

$T_{\text{iter}} \leftarrow (T_{\text{iter}}, T_{\text{temp}})$

}

Return $(T) = \{T_1, T_2, \ldots, T_{N_{\text{iter}}}\}$

End

4. IMPLEMENTATION FOR SEQUENTIAL CIRCUIT TEST GENERATION

For the generation of the test vectors for the given faults in the implementation, we used the program SIS, developed and maintained by Logic Synthesis Group at University of California, Berkeley. The fault simulation is also done using slightly modified SIS subroutines.

Fault simulation is an essential part of the testing method. It is the process of comparing the transitions and outputs of a fault free circuit and a circuit that contains one or more faults by simulating the functioning of such

circuits. For a given sequence of vectors we compare the output generated of the fault-free circuit to the circuit containing one fault at a time. Any fault that produces different output from the one of the fault-free circuit is thus detected by this sequence of vectors.

As input to our heuristics we supply the list of potential faults as the set of elements to cover and their JED-paths that detect them. For each JED- and ED-path delivered by SIS we apply fault simulation to identify the entire collection of faults identified by that vector sequence. This collection typically includes many more faults than the one for which the sequence was generated.

The initial state is set to the reset state of the circuit. Then the subtours produced by the heuristics are subtours in the STG that define sequences of input vectors to apply to the circuit in order to detect the faults.

When implementing our heuristics, an important point at which testing differs from the Tour Covering model is the *veering off* phenomenon. If a fault that causes an incorrect next-state transition is present in the circuit, the heuristics may not design valid tests. Consider for example a pair of faults $\{a, b\}$ that are present so that in the presence of a the ED-path devised for b veers off and still produces the correct output (while being at the wrong state). Similarly, the ED-path devised for a veers off in the presence of b and still produces the correct output. Consequently, with both these faults present our tests are not valid; the circuit appears to be free of these faults, but it is not. This error results from the generation of the JED-paths based on the assumption that only a single fault is present.

To protect against veering off we apply fault simulation (Williams and Parker 1983) to a tour, while checking that in the presence on any potential fault there is no deviation from the correct state transition.

The alternative to fault simulation to check for veering off is to check after every state transition that the tested (physical) circuit is in the correct state. Performing such a check is a costly process that reduces the speed and efficiency of the overall testing procedure. By comparison, the fault simulation is done in the procedure that devises the overall test, but does not affect the time required for the actual testing.

The fault simulation tends to be more conservative in that it may determine that certain tests are invalid, but in practice they are valid because the fault that causes the veering off is not present. As a result we may reject good sequences or subtours that can perform a valid test for a large collection of circuits. Nevertheless, in our empirical experience we observed that the tours generated never veered, and hence no subtour was rejected due to veering off.

For the Chinese Postman heuristic, all subtours produced in each iteration are fault simulated, so that the number of faults actually detected by the subtour is known before the best subtour is selected based on this information. With the Nearest Neighbor heuristic, before each sequence is added to the tour the sequence is fault simulated to check that it detects the fault. Any other faults it detects are removed from the fault list at this time. The Greedy Heuristic fault simulates each tour after it is constructed, also using *Fault Simulation* for the union of the faults covered to check that each fault which should be detected by the tour is indeed detected. Also, as with the Nearest Neighbor approach, any additional faults that the tour covers are removed from the fault list.

In the greedy heuristic we can control the length of the tour generated. As mentioned before, the density of detected faults—which is the average number of faults detected per input vector—tends to be very high initially, and even a short sequence of input vectors can cover a significant number of faults. After a test is found for a collection of faults, these checked faults can be removed from consideration. Thus the number of additional faults detected by later sequences decreases as more faults are detected and removed from the set, and the density goes down. For this reason we set for the Greedy Heuristic that initially it constructs tours from a small number of sequences. The length of the tours in terms of the number of sequences it contains is increasing as more tours are generated and more faults are detected. We chose to form the ith tour using i^2 sequences, thus setting N_i equal to i^2 in the Greedy Heuristic. As a result the tours produced initially, when the risk of veering off in the presence of almost all faults is high, tend to be short; whereas later on, when the number of remaining faults is small, we can afford to use very long tours without substantial risk of veering off.

In all our heuristics it is necessary to find shortest paths between states in the STG. Our code employs a breadth-first search to find these shortest paths, as edges in the STG are of unit length. For very large circuits our implementation does not enumerate the entire STG, due to memory and CPU time constraints. In this case paths found by breadth-first search are not necessarily the shortest ones. For all the circuits used in the current study, which are of small size, complete graphs were generated.

Another issue concerns the transportation problem used in the CPP Heuristic for Tour Covering. To solve this problem, we used a capacity scaling minimum cost flow algorithm described by Ahuja et al. (1992). Such an algorithm, employing capacity scaling, is appropriate for our needs since it can efficiently handle networks in which the nodes have small imbalances, compared to the number of nodes in the graph.

Another implementation detail involves constructing subtours in the Chinese Postman heuristic. If any subtour included in the final test does not go through the reset state, it is necessary either to patch this tour to another subtour which does pass through reset, or to link this subtour directly to the reset state. After observing that paths connecting a subtour to reset typically had unit length, we simply linked each subtour directly to reset, when reset was not already included in the subtour.

5. EMPIRICAL RESULTS

For the empirical study we first conducted runs using all three heuristics on the ISCAS-89 benchmark circuits (Brglez et al. 1989). The results from these runs are compared with runs on the same circuits using the single-fault and random approaches. Initial runs using the Chinese Postman and Nearest Neighbor heuristics were conducted when just one ED-path per fault was available. Additional runs considering more than one excitation state and differentiation sequence per fault were also done using these heuristics. Results from both are presented.

Each table includes the number of input vectors each heuristic used and the number of faults it covered. The number of testable faults in the circuit, which is the number of faults less the number of redundant faults, is also given, as well as the percentage of testable faults the heuristic detected. Also included is the ratio of faults covered per input vector. When the random approach required more than 5000 input vectors, the table records 5000+ input vectors, and the ratio is omitted. Results from the Chinese Postman heuristic are labeled as *CP*, those from the single-fault approach as *SF*, those from the random approach as *RN*, those from the Greedy Heuristic as *GR*, and those from the Nearest Neighbor heuristic as *NN*. Runs for the Chinese Postman and Nearest Neighbor heuristics that considered one ED-path are labeled as *sing*; those considering more than one are labeled as *mult*. The results of these runs are presented in Table I.

We also experimented with combining the heuristics in a multiple-stage approach to test generation. These results are presented in Table II. First we investigated using three stages—one for each new heuristic—and conducted two sets of runs, the first set using the Greedy Heuristic in the first phase, the Chinese Postman heuristic in Phase II, and the Nearest Neighbor heuristic in the last phase. The second set used the Nearest Neighbor approach first, then the Chinese Postman, with the Greedy Heuristic last. This ordering was inferior in terms of the quality of the results to the ordering using the Greedy Heuristic first. We also experimented using just the Greedy and Nearest Neighbor (a two-stage approach) and got some better tests using this approach. Results from these studies are presented in the next three tables. *NN* indicates the Nearest Neighbor heuristic and *CP* the Chinese Postman heuristic.

We chose to use either the Greedy or Nearest Neighbor heuristic in the first phase since it is important that initial tours do not have a large number of input vectors, as just a small number detect a considerable percentage of the faults. The Greedy Heuristic is then appropriate for the first phase since it constructs short initial tours. With the Nearest Neighbor heuristic we can limit the length of a tour, cutting it off after a few sequences are added, thus keeping it short. The rest of the faults can then be tested in another tour, which starts at reset. We also observed that after most of the faults have already been detected, the Chinese Postman heuristic tends to generate only one subtour that detects all of the remaining faults a multiple number of times, and therefore at high cost. For this reason we chose not to use the Chinese Postman heuristic in the third phase.

Two sets of runs were performed with each circuit. In the first set, tours were constructed in Phase I until at least 30% of the faults were detected. (A new tour was constructed in entirety in Phase I if less than 30% had been detected at that point.) In the second set of runs, tours were found in Phase I until at least 40% were detected. When the first phase used the Greedy Heuristic, the *first iteration alone* often detected more than 40% of the faults. Likewise, when Nearest Neighbor was run in the first phase, the *first vector alone* often detected more than 40% of the faults. In this case the two sets of runs show the same percentage of faults detected in the first phase. For both sets of runs, after the Chinese Postman Heuristic detected 30% of the faults in the second phase, test generation switched to the third phase. In cases where this would result in both runs producing the same set of tours, the first run switched to the third phase after detecting less than 30%, in order that the two runs produced different sets of tours. The percentage of faults actually detected by the Chinese Postman heuristic varied from circuit to circuit, depending on the subtours produced.

Next we experimented with a two-stage approach to test generation, using just the Greedy and Nearest Neighbor heuristics. The first set of runs used the Greedy Heuristic in the first phase, while the second set considered the Nearest Neighbor heuristic first. The results for the first ordering is presented in Table III.

In the column showing the number of input vectors used, asterisks appear next to an entry when the heuristic used the least number of input vectors among all heuristics for this circuit. Likewise, asterisks appear in the column showing the ratio of faults covered to number of input vectors when this ratio was greatest among all ratios for this circuit.

The Three-stage Approach using the Greedy Heuristic in the first phase gave the best results for nine of the thirteen circuits, the Two-stage Approach for two circuits, including one tie with the Three-stage Approach, and each pure heuristic gave the best result for one circuit: the Greedy Heuristic for circuit s386; the Chinese Postman heuristic for circuit cse; and the multiple excitation state Nearest Neighbor heuristic for circuit s1238. In every case, the heuristic with the best performance in terms of the number of input vectors used also had the best ratio of faults covered per input vector for the circuit.

The success of the Three-stage Heuristic indicates that it is worthwhile to combine heuristics in a multiple-stage approach to test generation. Such a strategy takes advantage of the fact that many faults are easy to detect, covering them as inexpensively as possible at the start. In addition it avoids constructing long, costly tours at the end, when most of the faults have already been covered, yet still applies optimization techniques in the second phase, setting it apart from a pure Nearest Neighbor or Greedy Heuristic.

Table I
Comparison of Various Approaches to Testing

Circuit	No. Gates	No. Testable Faults	Testing method	No. Input Vec's	No. Faults Cov'd	% Faults Cov'd	Faults Cov'd/Vec
s27	10	30	CP mult	25	30	100.0	1.20
			CP sing	25	30	100.0	1.20
			SF	30	29	76.7	0.97
			RN	25	30	100.0	1.20
			GR	20	30	100.0	1.50
			NN mult	15*	29	96.7	1.93
			NN sing	18	30	100.0	1.67
s208	96	175	CP mult	193	167	95.4	0.86
			CP sing	193	167	95.4	0.86
			SF	208	165	94.3	0.79
			RN	5000+	136	77.7	—
			GR	205	168	96.0	0.82
			NN mult	205	163	93.1	0.80
			NN sing	189	167	95.4	0.88
s298	119	273	CP mult	363	270	98.9	0.74
			CP sing	363	270	98.9	0.74
			SF	295	265	97.1	0.90
			RN	5000+	259	94.9	—
			GR	280	270	98.9	0.96
			NN mult	306	263	96.3	0.86
			NN sing	329	267	97.8	0.81
sse	130	298	CP mult	441	298	100.0	0.68
			CP sing	441	298	100.0	0.68
			SF	388	298	100.0	0.77
			RN	5000+	233	78.2	—
			GR	219	298	100.0	1.36
			NN mult	228	298	100.0	1.31
			NN sing	244	298	100.0	1.22
s386	159	314	CP mult	210	314	100.0	1.50
			CP sing	210	314	100.0	1.50
			SF	289	314	100.0	1.09
			RN	5000+	254	80.9	—
			GR	175*	314	100.0	1.79*
			NN mult	228	314	100.0	1.38
			NN sing	207	298	94.9	1.44
s344	161	319	CP mult	106	319	100.0	3.01
			CP sing	106	319	100.0	3.01
			SF	141	319	100.0	2.26
			RN	3100	319	100.0	0.10
			GR	101	318	99.7	3.15
			NN mult	110	318	99.7	2.89
			NN sing	115	318	99.7	2.76
s349	161	325	CP mult	106	325	100.0	3.07
			CP sing	106	325	100.0	3.07
			SF	136	325	100.0	2.39
			RN	3100	319	98.2	0.10
			GR	101	324	99.7	3.21
			NN mult	165	319	98.2	1.93
			NN sing	110	325	100.0	2.95
cse	192	517	CP mult	280*	517	100.0	1.85*
			CP sing	280*	517	100.0	1.85*
			SF	480	517	100.0	1.08
			RN	5000+	352	68.1	—
			GR	308	517	100.0	1.68
			NN mult	349	517	100.0	1.48
			NN sing	335	516	99.8	1.54

Continued

Since both the multiple-stage runs and the pure heuristics that were successful considered more than one excitation state per fault, it seems worthwhile to invest the computer time to locate additional excitation states and differentiation sequences for faults before constructing test sequences. Note that the random approach used more

Table I
Continued

Circuit	No. Gates	No. Testable Faults	Testing method	No. Input Vec's	No. Faults Cov'd	% Faults Cov'd	Faults Cov'd/Vec
s1238	508	1287	CP mult	623	1275	99.1	2.05
			CP sing	623	1275	99.1	2.05
			SF	809	1273	98.9	1.57
			RN	5000+	1207	93.8	—
			GR	546	1277	99.2	2.34
			NN mult	404*	1273	98.9	3.15*
			NN sing	557	1276	99.1	2.29
s1196	529	1242	CP mult	570	1233	99.3	2.16
			CP sing	570	1233	99.3	2.16
			SF	775	1228	98.9	1.58
			RN	5000+	1172	94.4	—
			GR	513	1233	99.3	2.40
			NN mult	393	1228	98.9	3.12
			NN sing	505	1234	99.4	2.44
sand	555	1330	CP mult	490	1300	97.7	2.65
			CP sing	490	1300	97.7	2.65
			SF	904	1301	97.8	1.44
			RN	5000+	1282	96.4	—
			GR	467	1301	97.8	2.78
			NN mult	502	1294	97.3	2.58
			NN sing	528	1294	97.3	2.45
planet	606	1431	CP mult	2513	1431	100.0	0.57
			CP sing	2513	1431	100.0	0.57
			SF	1548	1431	100.0	0.92
			RN	5000+	1428	99.8	—
			GR	628	1431	100.0	2.28
			NN mult	563	1426	99.6	2.53
			NN sing	626	1430	99.9	2.28
scf	959	2333	CP mult	2262	2304	98.8	1.02
			CP sing	2262	2304	98.8	1.02
			SF	2564	2304	98.8	0.90
			RN	5000+	677	29.0	—
			GR	2168	2303	98.7	1.06
			NN mult	2433	2307	98.9	0.95
			NN sing	2677	2294	98.3	0.86

than 5000 input vectors for all but three of the circuits, and for all but the first, very small circuit, required more vectors than any other approach.

We observed that many states function as an excitation state for more than one fault, or function both as an excitation state for one fault and as the end state of a differentiation sequence for another fault. Such overlapping makes it possible and even convenient to continue to append the test for the next fault to the end of the test for the current fault, indicating that constructing tours detecting more than one fault is a worthwhile approach. The results confirm this observation, as our heuristics indeed show a significant improvement over the single-fault approach.

Notice that our heuristics allow the flexibility of charging for different transitions different costs. In particular, it may be of interest to allocate an additional charge to the reset transition.

6. CONCLUSIONS AND FUTURE WORK

We have presented new approaches to test generation, improving upon past efforts. When applied to sequential circuits, our heuristics construct sequences of input vectors that efficiently check for the presence of circuit defects. Further, our methods are useful in that they exploit the knowledge of the physical plan of the circuit, and are thus more applicable to circuit manufacturing environments.

In expanding the work we plan to incorporate a new procedure for generating test vectors for faults at the third phase. The purpose is to create full coverage of all faults. At the start of the third phase, after our heuristics have generated tests for most of the faults, it is often difficult to find short tests for the remaining faults. We may need to include additional features in our heuristics in order to test for this last batch of faults while still maintaining low average density of vectors per fault. One such procedure we consider is the technique of Product Machine Traversals reported by Cho et al. (1991).

We plan to investigate further the cutoff point for switching between the heuristics. A preliminary study has shown that the results could be further improved by choosing the percentage of faults covered by each heuristic

Table II
Three Stage Approach

	Phase I: Greedy		Phase II: CP		Phase III: NN		Summary		
Circuit	% faults cov'd	No. input vec's	% faults cov'd	No. input vec's	% faults cov'd	No. input vec's	% faults cov'd	No. input vec's	Faults cov'd/vec
s27	33.3	3	36.7	3	30.0	9	100.0	15*	2.00*
s27	90.0	12	10.0	6	0.0	0	100.0	18	1.67
s208	36.6	7	30.8	14	28.6	152	96.0	173*	0.97*
s208	80.0	51	16.0	176	0.0	0	96.0	227	0.74
s298	57.1	13	33.0	33	7.3	195	97.4	241*	1.10*
s298	57.1	13	29.3	13	11.0	257	97.4	283	0.94
sse	44.6	8	55.4	520	0.0	0	100.0	528	0.56
sse	44.6	8	16.4	9	38.9	199	99.9	216*	1.38*
s386	40.8	7	31.2	24	27.7	188	99.7	219	1.43
s386	40.8	7	23.2	15	35.7	199	99.7	221	1.42
s344	83.4	29	7.2	8	9.4	80	100.0	117	2.68
s344	83.4	29	15.0	27	1.6	10	100.0	66*	4.76*
s349	67.4	7	31.1	40	1.2	17	99.7	64*	5.06*
s349	83.1	29	10.5	16	6.5	61	100.0	106	3.07
cse	65.2	57	31.5	179	3.3	72	100.0	308	1.68
cse	65.2	57	34.8	259	0.0	0	100.0	316	1.64
s1238	41.4	35	30.2	48	27.5	375	99.1	458	2.78
s1238	41.4	35	30.2	48	27.5	375	99.1	458	2.78
s1196	78.7	210	1.6	2	18.8	217	99.1	429	2.89
s1196	78.7	210	2.5	4	18.0	215	99.2	429	2.89
sand	32.8	13	31.6	44	32.7	418	97.1	475	2.72
sand	62.2	61	33.7	286	2.2	88	98.1	435*	3.00*
planet	45.0	17	10.6	11	44.2	494	99.8	522*	2.74*
planet	45.0	17	55.0	2506	0.0	0	100.0	2523	0.57
scf	44.4	86	31.9	87	22.6	1974	98.9	2147*	1.07*
scf	44.4	86	31.9	87	22.6	1974	98.9	2147*	1.07*

Table III
Two-stage Approach

	Phase I: Greedy		Phase II: NN		Summary		
Circuit	% faults cov'd	No. input vec's	% faults cov'd	No. input vec's	% faults cov'd	No. input vec's	Faults cov'd/vec
s27	90.0	12	10.0	3	100.0	15*	2.00*
s208	80.0	51	16.0	128	96.0	179	0.94
s298	57.1	13	39.6	297	96.7	310	0.85
sse	57.7	27	42.3	196	100.0	223	1.34
s386	57.6	31	42.0	200	99.6	231	1.35
s344	67.4	7	32.3	115	99.7	122	2.61
s349	67.4	7	32.0	109	99.4	116	2.78
cse	65.2	57	34.8	275	100.0	332	1.56
s1238	57.2	73	42.1	396	99.3	469	2.72
s1196	57.8	70	41.3	320	99.1	390*	3.16*
sand	62.2	61	36.1	448	98.3	509	2.57
planet	61.5	40	38.3	518	99.8	558	2.56
scf	61.6	231	37.4	2063	99.0	2294	1.01

properly. In order to have conclusive results, the experiments will be run on very large circuits.

After researching these ideas, we plan to apply our method to real size circuits and investigate modifications that will be particularly applicable for such circuits.

An approach proposed but not fully investigated is the set covering/crew scheduling model for testing. The flexibility of crew scheduling problems in handling paths in the network render this model particularly attractive and promising for future research.

ACKNOWLEDGMENTS

The authors wish to thank Alexander Saldanha for important discussions which led to the initial development of this project. We also express much gratitude to Sushil Verma

for his significant contribution to this project, not only in software development but in getting the project off the ground and in experimenting with new heuristics. We would like to thank Logic Synthesis Group at UC Berkeley for providing us access to SIS. Questions about SIS and its origin should be sent to sis@ic.eecs.berkeley.edu.

The first author was supported by ONR, National Defense Science and Engineering Graduate fellowship. The second author was supported in part by ONR under grant N00014-91-J-1241 and by the Competitive Semiconductor Manufacturing project by the Sloan Foundation.

REFERENCES

AHO, A. V., A. T. DAHBURA, D. LEE, AND M. U. UYAR. 1982. An Optimization Technique for Protocol Conformance Test Generation Based on UIO Sequences and Rural Chinese Postman Tours. *Proc. IFIP WG 6.1 Eighth Intl. Symp. on Protocol Specification, Testing, and Verification.* Atlantic City, NJ, 75–86.

AHUJA, R. K., T. L. MAGNANTI, AND J. B. ORLIN. 1992. *Network Flows: Theory, Algorithms, and Applications.* Prentice-Hall, Englewood Cliffs, NJ.

ANBIL, R., C. BARNHART, AND E. L. JOHNSON. 1991. A Column Generation Technique for the Long-Haul Crew Assignment Problem. Manuscript.

BRGLEZ, F., D. BRYAN, AND K. KOZMINSKI. 1989. Combinational Profiles of Sequential Benchmark Circuits. *Proceedings of the International Symposium on Circuits and Systems*, Portland, Oregon.

CHO, H., G. HACHTEL, AND F. SOMENZI. 1991. *Fast Sequential ATPG Based on Implicit State Enumeration*. International Test Conference.

EDMONDS, J. AND E. L. JOHNSON. 1973. Matching Tours and the Chinese Postman. *Math. Programming,* **5,** 88–114.

FÜREDI, Z. AND R. P. KURSHAN. 1987. Minimal Length Test Vectors for Multiple-Fault Detection with Electron Beam Scanning. Manuscript.

GAREY, M. R. AND D. S. JOHNSON. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, NY.

GHOSH, A., S. DEVADAS, AND A. R. NEWTON. 1991. Test Generation and Verification for Highly Sequential Circuits. *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems,* **10,**5 652–667.

GRIMALDI, R. P. 1989. *Discrete and Combinatorial Mathematics: An Applied Introduction*. Addison-Wesley, Reading, MA.

HU, T. C. 1982. *Combinatorial Algorithms*. Addison-Wesley Publishing Company, Reading, MA.

LAWLER, E. 1976. *Combinatorial Optimization Networks and Matroids*. Saunders College Publishing, Fort Worth, TX, 260–261.

LEE, D. AND M. YANNAKAKIS. 1992. Testing Finite State Machines: Fault Detection. Manuscript.

MANO, M. M. 1991. *Digital Design*. Prentice-Hall, Englewood Cliffs, NJ.

PAPADIMITRIOU, C. H. AND K. STEIGLITZ. 1982. *Combinatorial Optimization and Complexity*. Prentice-Hall, Inc., Englewood Cliffs, NJ.

PIXLEY, C., S. JEONG, AND G. HACHTEL. 1992. *Exact Calculation of Synchronization Sequences Based on Binary Decision Diagrams*. 29th ACM/IEEE Design Automation Conference, 620–623.

WILLIAMS, T. W. AND K. P. PARKER. 1983. Design for Testability—A Survey. *Proc. IEEE*, **71,** 98–112.