Reformulation of the Generation of Conformance Testing Sequences to the Asymmetric Travelling Salesman Problem

Jitian Xiao, Chiou Peng Lam, Huaizhong Li, Jun Wang*
School of Computer and Information Science, Edith Cowan University,
2 Bradford Street, Mt Lawley, WA 6050, Australia
{j.xiao, c.lam, h.li, j.wang}@ecu.edu.au

ABSTRACT

Protocol conformance testing generally involves checking whether the protocol under test conforms to the given specification. An important issue in protocol conformance testing is the generation of test sequences in an efficient and effective way that achieves the required fault detection coverage. This paper proposed an approach for finding the shorter test sequences for protocol conformance testing based on the Wp method. The approach provides a technique for transforming the problem of the test sequence generation from a given FSM into one of finding the shortest path in the asymmetric travelling salesman problem by using one of the many existing meta-heuristic algorithms for addressing TSP. The approach addresses the issue of reformulation of Software Engineering problems as search-based problems in Search-based Software Engineering. The paper also shows that the resulting test sequences will maintain the same fault detection capability as those of the Wp method.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification - *Formal methods, reliability*.

General Terms: Algorithm, Verification, Reliability.

Keywords: Conformance testing, test sequence, Wp method, TSP.

1. INTRODUCTION

Many systems such as communication protocols and control systems can be modelled using the finite state machines (FSMs). Protocol conformance testing generally involves checking whether the implementation under test conforms to a given specification. A sequence of inputs, generated from the specification, is applied to the implementation and the outputs is verified as to whether the expected sequence of outputs is obtained [1][3]. As the implementation is a black box from a testing perspective, the protocol conformance testing problem is difficult to tackle efficiently [1]. One of the most important issues in protocol conformance testing is how to generate the sequence of inputs, called the test sequence, in an efficient and effective way to achieve the required fault detection coverage.

Many methods have been proposed to address the protocol conformance testing problem [1][2][3][6]. Among these, the W-method [4] and the improved Wp method [5] have attracted much attention. Unlike other methods, the W-method and Wp method can be applied to all protocols, and can guarantee the detection of any output and transfer faults under certain conditions [4][5]. However, these two methods assumed that the implementation under test has reliable reset functions which is difficult to be realized for some systems. If an implementation doesn't have a reliable reset function, alternative reset technique can be used to home the test sequence [5]. In addition, the length of the generated test sequences using these two methods are usually the longest amongst all methods. Attempts to optimise the generated test sequences based on the Wp method (i.e. shorter test sequences) include the use of the Rural Chinese Postman (RCP) algorithm in a technique proposed in [3].

This paper proposed an approach for finding shorter test sequences for protocol conformance testing based on the Wp method. The approach provides a transformation technique for converting the problem of the test sequence generation from a given FSM into one of finding the shortest path in the asymmetric travelling salesman problem (ATSP) using one of the many existing meta-heuristic algorithms for addressing TSP. This paper, like some papers in search-based software engineering, reformulates a software engineering problem, (i.e. state machine testing), as a search-based problem (i.e. the asymmetric TSP). An important issue, as indicated in the discussion section with the proposed approach, is that the resulting test sequences will maintain the same fault detection capability as those of the Wp method when protocol conformance testing is concerned.

2. RELATED WORK

Many approaches, such as Transition Tours (*T*-method), Distinguishing Sequence (*DS*-method), Characterising Sequences (*W, Wp method*) and Unique Input/Output Sequences (*UIO, UIOv*-method) have been proposed previously to address the conformance testing problem (see, for example, [1][2][3][5][6]). It has been shown previously that the T-method does not have good fault detection capability and very few protocols possess DS. The UIO is also not applicable to all protocols; only the W-method and Wp method can be applied to all protocols. As mentioned previously, if there is no reliable reset function in the FSMs, the test sequences

*This author works in the School of Computer Science and Computer Engineering, Wenzhou University, China. He was a visiting fellow at Edith Cowan University when the paper was written.

generated using the Wp method and the UIO method are generally very long.

The UIO method was improved in [1] where the RCP technique was proposed to replace the reset function whereby test segments are connected into a continuous test sequence. The RCP approach was also applied in [3] to the Wp method to produce an optimal test sequence which includes all non-overlapping test segments. Other attempts to produce optimal test sequences include some initial attempts of [8][9] where the problem is first transformed and mapped to an ATSP problem, followed by the applications of the Hopfield Neural Network [9] or simulated annealing [8] technique respectively. However, the work in both [8] and [9] are based on multiple UIO test sequences.

Chen's approach [3] involved generating test segments from the Wp method as the first step. The second step checks if any pair of the test segments from the first step can be overlapped. If so, a new test segment is then constructed from the pair of overlapped test segments. Lastly, RCP is used to connect the newly formed test segments of step 2 in order to generate a minimum length test sequence. The work in [8] and [9] first generate multiple UIO test sequences. The node, n_i of a TSP used here corresponded to a set of sub-sequences that mapped to e_i, one of the edges of the protocol.

A key issue in Search-based Software Engineering is the need to recast software engineering problems into search-based problems. In existing work involving structural testing, the problem of test data generation is recast as a problem of searching for test inputs which will satisfy a specified test adequacy criterion. For example, Harman *et al* presented a testability transform that transform a code based program with one or more *exit* statements into a structured branch-covering equivalent program [11]. In the area of project planning in software maintenance, Antoniol *et al* [10], recast the problem as one of scheduling. While there is some very initial work involving transformation of test sequence generation in conformance protocol testing, none has involved transforming the problem of the test sequence generation based on the Wp method into one of finding the shortest path in the asymmetric travelling salesman problem. Furthermore, it needs to be emphasized that although the resultant test sequences were claimed as 'optimal' in [1][3][8][9], the optimality was achieved in comparison to the original UIO and Wp test sequences respectively. There is no guarantee that the shortest test sequence can be obtained using these methods.

The proposed approach processes the test segments generated from the Wp method to remove redundancy before generating the resulting ATSP. This generally leads to shorter test sequences being generated. This also reduces the complexity of the resulting ATSP, ensuring that the computations involved with the test sequence generation are greatly reduced.

TSP belongs to the class of computationally hard problems. The problem involves finding the optimal (shortest) length of a tour where a salesman starts at an initial city and visits all n cities in the tour exactly once. Usually in a TSP, the distance between 2 cities is the same in both directions but in an ATSP, the distances between two cities can differ depending on the direction traveled. There are many existing algorithms developed to address this class of problem. These techniques can be divided into two categories: exact algorithms that finds exact solutions but only work reasonably well when the size of the problem is small (e.g. *Branch and Bound* algorithm for TSP with 40-60 cities and *Linear Search* method involving TSP with 120-200 cities) [13] and heuristics which will find "good solutions with high probability (e.g. *Nearest Neighbour* approach, *k-opt, Simulated Annealing*, ant system, genetic algorithms and *Neural Network*).

3. PRELIMINARIES

A deterministic finite state machine (FSM) M is a quintuple (S, X, Y, δ, λ) where S is the finite set of states which includes the special state s_0 called the *initial state,* X is the finite set of inputs, Y is the finite set of outputs which includes "null", δ: $S\times X \rightarrow S$ is the transfer function, λ: $S\times X \rightarrow Y$ is the output function [6].

If functions δ and λ are defined for all $(s_i, x_j) \in S\times X$, the FSM M is called *completely specified*. It is always possible to render a FSM completely specified [2]. Therefore, without loss of generality, we assume that the considered FSMs are completely specified.

The transfer function δ and the output function λ can be combined to form a transition relation t. For $(s_i, x_j) \in S\times X$, if $\delta(s_i, x_j)= s_k$ and $\lambda(s_i, x_j)= y_h$, a transition $t(s_i, x_j, s_k, y_h)$ in the transition set T is defined as t: $(s_i, x_j) \rightarrow (s_k, y_h)$. If M is completely specified, $|S|=n$, and $|X|=m$, then $|T|=n\times m$.

A labelled digraph G [1] is denoted as G = (V, E) where V is a set of vertices and E is a set of labelled directed edges which link the vertices. An edge e which starts from v_i and ends at v_j with distinct label *l* is denoted by $e(v_i, v_j, l)$.

If V=S, and E includes an edge $e(s_i, s_k, x_j/y_h)$ if and only if T includes transition $t(s_i, x_j, s_k, y_h)$, then a digraph G(V, E) is a *derived digraph* of the corresponding FSM M. Due to the equivalence of e and t, in the following we will not distinguish between e and t, and will subsequently denote the derived digraph of M as G(S, T), sometimes called *transition digraph*. A transition digraph of an FSM M is given in Figure 1.

The function δ, function λ and transition t can be extended to input sequences. For completely specified FSMs, for a given state s_i and a given input sequence $\bar{x}=\{x_1, x_2, ..., x_n\}$, there is a transition sequence: $t_1(s_i, x_1, s_{i+1}, y_1)$, $t_2(s_{i+1}, x_2, s_{i+2}, y_2)$, ..., $t_n(s_{i+n-1}, x_n, s_{i+n}, y_n)$. We denote the transition sequence as $\bar{t}(s_i, \bar{x}) = \{t_1, t_2, ..., t_n\}$. On the other hand, given a transition sequence $\bar{t}$, there is an input sequence $\bar{x}$ which corresponds to $\bar{t}$. Therefore, for simplicity and without loss of generality, we frequently use $\bar{t}$ in the place of $\bar{x}$. Similarly, we denote the output sequence $\bar{y} = \lambda(s_i, \bar{x})=\{y_1, y_2, ..., y_n\}$ for a transfer $\delta(s_i, \bar{x})=s_{i+n}$. Note that the extension of the definitions of δ and λ in this paper is slightly different from that in [6]. In the derived digraph G(S, T), there exists an adjacent edge sequence $\bar{t}(s_i, \bar{x}) =\{ (s_i, s_{i+1}, x_1/y_1), (s_{i+1}, s_{i+2}, x_2/y_2),..., (s_{i+n-1}, s_{i+n}, x_n/y_n)\}$ corresponding to the transfer from s_i to s_{i+n}. We call $\bar{t}$ a *walk* of G from s_i to s_{i+n}, while s_i is the starting state and s_{i+n} the ending state of $\bar{t}$.

An FSM is *strongly connected* if there exists a *walk* between any pair of two distinct states s_i and s_j in its derived digraph. In this paper, we assume that FSM is *strongly connected* which is also a standard assumption for approaches based on the Wp method. Strongly connected FSMs have complete reachability, i.e., there is always a feasible walk between two distinct states in the derived digraph.

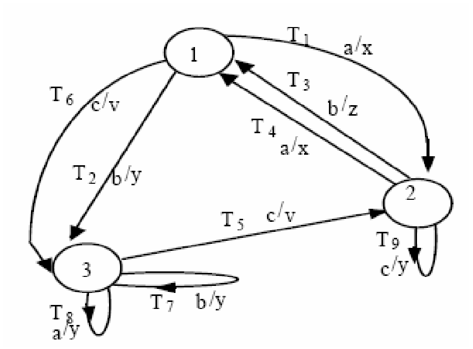

Figure 1. A transition digraph G(V, E) of M [3]

Two states in two FSMs are *equivalent* if they produce the same output sequence for any input sequence. Two FSMs are *equivalent* if and only if for every state in one FSM, there is an equivalent state in the other FSM. An FSM M is *minimal* if all other equivalent FSMs have equal or more number of states than M. An FSM M is minimal if and only if there is no equivalent state in M [6].

Suppose that we are given a system which is specified by an FSM M, and its implementation-under-test (IUT) is another FSM I, the problem to determine whether I is equivalent to M is called "conformance testing" or "fault detection" problem. Similar to [3][6], we assume that the specification FSM M and its implementation FSM I are all minimal.A test sequence of the Wp method can be partitioned into three parts: the prefix segment, the transition to be checked and the suffix segment. The prefix segment is an element of a state cover set Q which starts from the initial state s_0 and ends at the starting state s_i of the transition to be checked. Supposing that the transition ends at state s_j, the suffix segment of the test sequence is one of the elements of the state identification set W_j[1]. For example, in Figure 2, the state identification set for s_4 is {(T5, T6), (T7)}}.

In order to detect the transition T4, two test sequences have to be generated: (T1,T2,T3,T4,T5,T6) and (T1,T2,T3,T4,T7). The prefix segment is (T1,T2,T3), and the suffix segments are (T5,T6) and (T7) respectively. The transition to be tested together with the suffix segment is called the "*test segment*". In Figure 2, [T4,T5,T6] and [T4,T7] are two test segments to detect transition T4 (denoted by Check(T4)[1] and Check(T4)[2], respectively, in [3]). Note that we have used the transition sequence to represent the corresponding input sequence here.

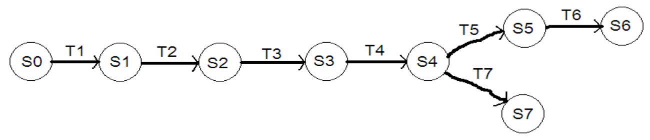

Figure 2. Test sequences for a transition

[1] An identification set of a state s_j is a transition set W_j that has the property that, for each state s_k, $s_k \neq s_j$ implies $\lambda(s_k, W_j) \neq \lambda(s_j, W_j)$.

4. AN OPTIMIZED TECHNIQUE TO GENERATE TEST SEQUENCE BASED ON THE WP METHOD

In this section, we describe our approach which will further improve the results already reported in [3].

4.1 Problem Definition and Motivation

For a given transition digraph G of an FSM, we assume that there are *n* test segments $S_1, S_2, \ldots, S_n$, each consisting of a set of transitions

$$S_i = (T_{i_1}, T_{i_2}, \ldots, T_{i_k}),\ k \geq 2. \tag{1}$$

According to Chen [3], if all test segments are to be tested separately in a final test sequence π, then π should appear in a form of

$$\pi:\ S_{\pi_1}, L_1, S_{\pi_2}, L_2, \ldots, S_{\pi_{n-1}}, L_{n-1}, S_{\pi_n} \tag{2}$$

where $S_{\pi_1}, S_{\pi_2}, \ldots, S_{\pi_{n-1}}, S_{\pi_n}$ is a permutation of $S_1, S_2, \ldots, S_n$ (i.e., $\pi_k \neq \pi_j$ when $k \neq j$ for $1 \leq k,\ j \leq n$) and L_k is a path linking S_{π_k} and $S_{\pi_{k+1}}$ in G, $k=1, 2, \ldots, n-1$. The total length of such a test sequence (in number of transitions) is

$$\begin{aligned} C_\pi &= \sum_{k=1}^{n} length(S_{\pi_k}) + \sum_{k=1}^{n-1} length(L_k) \\ &= \sum_{k=1}^{n} length(S_k) + \sum_{k=1}^{n-1} length(L_k) \end{aligned} \tag{3}$$

Where $length(S_k)$ ($length(L_k)$, respectively) is the number of transitions in path S_k (L_k, respectively). $length(L_k)$ is called the *cost* of connecting S_{i_k} and $S_{i_{k+1}}$ in G. Let

$$s = \sum_{k=1}^{n} length(S_k) \tag{4}$$

and

$$l = \sum_{k=1}^{n-1} length(L_k) \tag{5}$$

Then $C_\pi = s + l$. We call l the (total) cost of connecting test segments in G. For a given FSM derived graph, s is a constant.

The goal of test sequence generation is to find a test sequence π such that C_π as defined by formulae (3), (4) and (5) is minimized. Chen's work [3] tried to minimize l only. However it didn't do any reduction on s.

4.2 The New Approach

The generation of the shortest possible test sequence in the context of this study is based upon Wp method. The problem of transformation of the test sequence generation problem to ATSP, followed by the generation of the TSP tour and final test sequences involves four phases.

4.2.1 Phase 1: Generating Test Segments

For a given FSM M and its derived graph G, we produce a set of test segments using the Wp method. Due to space limitations, details relating to the generation of test segments using the Wp method are omitted here. For a comprehensive example of this step, refer to Chen's work [3].

4.2.2 Phase 2: Removing Overlapping Test Segments

This phase reduces the number of the generated test segments (i.e., n of formula (3)) in the final test sequence. If a test segment is totally contained in another test segment (as a subsequence), we simply remove it from the test segment set. We call the resultant segment set the *simplified* test segment set, denoted by S.

4.2.3 Phase 3: Generating Segment Sequences

We concatenate a series of test segments if the end transition of one test segment is the same as the start transition of another by eliminating the overlapping start/end transitions. For example, if two test segments are [T_1, T_2] and [T_2, T_3], respectively, then a subsequence (T_1, T_2, T_3) in the final test sequence would cover both test segments and will be shorter than that connecting both test segments using any other links (for example, in a form of (2)).

In order to reduce the number of overlapping transitions between consecutive segments in the final test sequence, we now define an auxiliary weighted digraph, called *segment graph*, $G_1 = (V_1, E_1, w)$, to represent the relationships between transitions in the test segment set. If a test segment starts at transition T_1 and ends at T_2, then these two transitions are linked in G_1. Formally, V_1 consists of all testable transitions appearing in the simplified test segment set S. For each test segment $[T_{i1}, T_{i2}, \ldots, T_{ik}] \in S$, we define a directed edge $e = <T_{i1}, T_{ik}> \in E_1$. The weight of such an edge e is defined as the number of transitions in the corresponding test segment, i.e., $w(e)=k$. If two or more test segments start from the same starting transition T_i and end at the same ending state T_j, we define an edge in E_1 for each of these segments, and denote them $<T_i, T_j>^1$ and $<T_i, T_j>^2$, and so on. Intuitively, an edge in E_1 can be explained as a path that bridges transitions from T_i to T_j within the test segment $[T_i, \ldots, T_j]$.

Table 1. A set of test segments for the FSM M (from [3])

Starting State	Test Segments	Ending State
1	Check(T1)=[T1, T3]	1
1	Check(T2)=[T2, T8]	3
2	Check(T3)^1=[T3, T1]	2
2	Check(T3)^2=[T3, T2]	3
2	Check(T4)^1=[T4, T1]	2
2	Check(T4)^2=[T4, T2]	3
3	Check(T5)=[T5, T3]	1
1	Check(T6)=[T6, T8]	3

As an example, consider the set of test segments in Table 1 which is generated for the FSM shown in Figure 1. As [T1, T3] is a test segment (i.e., Check(T1)) in Table 1, by definition, there is a directed edge <T1, T3> in its segment graph G_1, and the weight of the edge, w_1(<T1, T3>) = 2 because there are two transitions in the test segment. Other edges of G_1 can be defined similarly, and we finally get a segment graph $G_1=(V_1, E_1, w_1)$ as shown in Figure 3, where $V_1= \{T_1, T_2, \ldots, T_8\}$; $E_1=\{<T_1, T_3>, <T_2, T_8>, <T_3, T_1>, <T_3, T_2>, <T_4, T_1>, <T_4, T_2>, <T_5, T_3>, <T_6, T_8>\}$. The weights are shown on the edges of the graph.

From the definition, a path consisting of k edges in G_1 corresponds to a sequence of k test segments in S such that the end transition of the ith test segment is the same to the start transition of the (i+1)th test segment, i=1, 2, …, k-1. These k test segments could be connected with no cost if we eliminate the overlapping part (i.e., the start transition of each test segment, except for the first segment, in the sequence) between them. If we connect these k segments as a sub test sequence and put it in the final test sequence, not only the total cost of connecting those segments in the sub sequence is reduced (thus reduced l in formula (5)), but the total length of the test segments in the sub sequence is also reduced (thus reduced s in formula (4)).

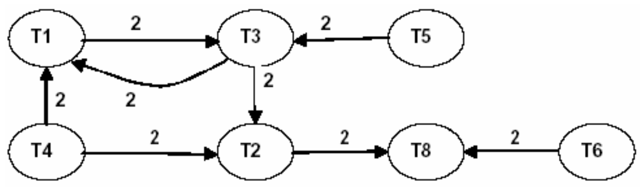

Figure 3. Segment graph G_1 for test segments of Table 1

As our goal is to find a shorter test sequence, we first concatenate as many test segments as possible if they do not need any linking transitions in the original FSM derived digraph, and subsequently replace the test segments with their concatenated result in the test segment set. In other words, we generate a minimum number of concatenated test segments such that every test segment is concatenated exactly once. It is evident that the generation of such concatenated test segments is equivalent to the problem of finding a minimum number of paths of G_1 that cover every edge exactly once. We call each of these resultant test segments in G_1 an *end-overlapping segment sequence (EOSS)*.

An EOSS, $S_{j_1}, S_{j_2}, \ldots, S_{j_k}$, k>1, has the following properties:

(i) S_{j_l} is a test segment, $1 \le l \le k$;

(ii) The ending transition of S_{j_l} is the same as the starting transition of $S_{j_{l+1}}$, $1 \le l \le k-1$.

Based on property (ii), for any EOSS $S_{j_1}, S_{j_2}, \ldots, S_{j_k}$ (k>1), we construct a sequence (S_{j_1} , *trailing*(S_{j_2}), …, *trailing*(S_{j_k})), called *segment sequence*, where *trailing*(S_{j_l}) is the sequence of transitions of S_{j_l} excluding the first transition of S_{j_l} , l=2, 3, …, k. The total length of such a segment sequence is:

$$\begin{aligned} & length(S_{j_1}) + \sum_{l=2}^{k} length(trailing(S_{j_l})) \\ &= length(S_{j_1}) + \sum_{l=2}^{k} (length(S_{j_l}) - 1) \\ &= \sum_{l=1}^{k} length(S_{j_l}) - (k-1) \end{aligned}$$

By taking these segment sequences as a sub test sequence, we reduce the length of the final test sequence by at least k-1, when comparing with s in formula (4).

The algorithm (in C-like pseudocode) used to generate the set of segment sequences is as below:

Algorithm SegSeq(G_1)
Input: segment graph $G_1=(V_1, E_1, w_1)$;
Output: segment sequences.

```
BEGIN
  For all nodes $n_i \in V_1$
      calculate indegree($n_i$) and outdegree($n_i$)[2];
  E=$E_1$;
  ss=1;
  While ($E \neq \emptyset$)
   {V=V1;
    print ("P", ss, ": ("); //new segment sequence
    ss++;
   Select a node $n_i \in V$ of minimal indegree() and outdegree($n_i$)>0
     {CurNode = $n_i$; Print CurNode;
       While there exists a node $n_j \in V$ such that <CurNode, $n_j$>$\in$E
         {Select a node $n' \in V$ such that
                   $w_1$(<CurNode, n'>)=$\max_j$($w_1$(CurNode, $n_j$>))
           {print n'; Outdegree(CurNode) --; indegree(n') --;
            Remove <CurNode, n'> from E;
            Remove CurNode from V;
            CurNode = n';
           };
        }; //end of while
     }; // end of select
    Print (") \n"); //Cur segment sequence ended.
   }; //end of While
END;
```

Generally, there may not exist any single segment sequence that goes through all edges of G_1 exactly once. In this case, we may produce multiple segment sequences such that every edge of G_1 is covered exactly once. For a given segment graph G_1, this strategy may result in multiple solutions of segment sequences[3]. For example, one such solution consisting of four segment sequences is shown below (in the order of total weight of the edges in each segment sequence) in Table 2 and Figure 4.

Table 2. Segment sequences for test segments in Table 1

Segment Sequence	appearing in Figure 4
(T4, T1, T3, T2, T8)	in thin solid arrowed lines
(T5, T3, T1)	in thin dotted arrowed lines
(T6, T8)	in thick solid arrowed lines
(T4, T2)	thick dotted arrowed lines

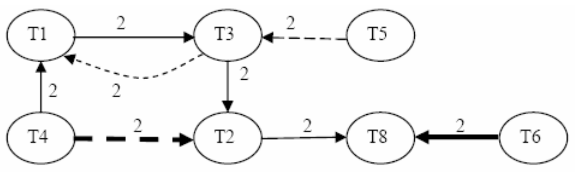

Figure 4. A segment graph G_1 and its segment sequences marked in different line types

[2] The number of edges which start from a node is called the *Outdegree* of the node. The number of edges which end at a node is called the *Indegree* of the node

[3] The problem of selecting optimal solutions of segment sequence set is beyond the scope of this paper and is thus not discussed in this paper.

4.2.4 Phase 4: ATSP and Final Test Sequence

A test sequence can be generated by linking the above generated segment sequences together. In order to produce a shorter test sequence which covers these segment sequences, the linking segments should be selected such that *l* in formula (4) is minimized. To do this, we construct another auxiliary weighted digraph $G_2 = (V_2, E_2, w_2)$ as below: Each segment sequence P= $(S_i, \ldots, S_j)$ forms a node in V_2. For any pair of segment sequences P1 and P2 ($P1 \neq P2$), there is a directed edge $<P1, P2> \in E_2$. Assume that the end state of P1 (i.e., the end state of the last transition in P1) is s_{P1} in the original FSM derived digraph G and the start state of P2 (i.e., the start state of the first transition in P2) is s_{P2} in G. Then due to the fact that G is strongly connected, the directed edge <P1, P2> always exists and its weight is defined as the number of transitions in the shortest path from s_{P1} to s_{P2} in G. In particular, if $s_{P1} = s_{P2}$, the weight of <P1, P2> is 0, indicating the two segment sequences P1 and P2 are linked end-by-end. The shortest paths between pairs of nodes in the original FSM derived digraph G that were used to link the segment sequences can be generated using, for example, Dijkstra's shortest path algorithm [12].

As each segment sequence corresponds to a node in G_2, the problem of finding the shortest test sequence which covers these segment sequences can now be converted to finding the shortest path from G_2 that goes through every node of G_2 exactly once. In other word, the problem can be converted to the well-known *travelling salesman problem*[5] (TSP) over G_2.

Note that a TSP tour must exist in G_2 because G_2 is also strongly connected, i.e., for any pair of nodes P1 and P2 ($P1 \neq P2$) in G_2, there is a directed edge $<P1, P2> \in E_2$. Once a TSP tour is found, by breaking the tour from an edge with the highest weight in the tour, we get a "shortest path"[6] that goes through every node of G_2 exactly once.

The final test sequence is then produced by taking all transitions along the shortest path, including those inside the nodes (i.e., the transitions in the corresponding segment sequences) and those on the edges (i.e., the transitions that link two consecutive segment sequences). The algorithm is informally stated using the following pseudocode:

```
Algorithm TestSeqGen(G)
  Input: FSM derived graph G;
  Output: test sequence of M.
  BEGIN
 1.  Generate the test segment set, S, using Wp method from G;
 2.  /*Remove redundant test segments*/
     For all pairs of test segments $S_i$ and $S_j \in S$
        If $S_i$ is totally contained in $S_j$
              remove $S_i$ from S;
 3.  Construct segment graph $G_1$; //see phase 3
```

[5] The *traveling salesman problem* is to find a minimum-cost tour of a given graph, covering every node exactly once [22].

[6] The path generated this way may not always make it the shortest TSP path, however, it is a practice heuristic to generate shortest TSP path.

```
4. Call SegSeq(G_1); /*generate the set of segment sequences
   from G_1, see phase 3 */
5. Calculate shortest paths between pairs of states from initial G
   (using Dijkstra' shortest path algorithm); //see phase 4
6. Construct auxiliary graph G_2 ; //see phase 4
7. Generate TSP tour from G_2; //see phase 4
8. /*generate final test sequence*/
   /*for simplicity and without lose of generality, suppose the
     generated TSP tour appears in <P_1, P_2, P_3,…, P_{m-1}, P_m>*/
   For i=1, 2, …, m-1 do
     {Output transitions in segment sequence P_i;
      Output transitions on the shortest path between the last
      state of P_i and the first transition of P_{i+1;}
     };
   Output transitions in segment sequence P_m;
END;
```

4.3 An Example

The example shown here is taken from [3]. Consider an FSM M represented by the transition digraph G which has been shown in Figure 1. All the assumptions used in [3] are also adopted here. Note that it was declared in [3] that only transitions T1, T2, …, T6 are required to be checked as these transitions represented the main behaviour of the protocol. For comparison purpose, we have also adopted this declaration.

A set of test segments for G as shown in Table 1 is obtained in [3]. It is reported in [3] that an optimal test sequence for G in Figure 1 has a total length of 23.

We now illustrate our proposed method of test sequence generation by going through the algorithm TestSeqGen. We use the FSM derived digraph G in Figure 1 as the specification FSM.

In Step 1, a set of test segments is generated based on the Wp method. The generated test segments are listed in Table 1 (This step is the same as that of Chen's method [3]).

Since no single test segment listed in Table 1 is totally contained in any other test segment, Step 2 does no change the test segment set. Thus S contains the same test segments as those found in Table 1.

In Step 3, a segment graph G_1 for this example is constructed as shown in Figure 3.

In Step 4, we produce the segment sequences that cover every edge of G_1 exactly once. Suppose that the execution of algorithm SegSeq( ) generated segment sequences as below (note that, depending on selection of the starting node, alternative solutions may exist),

P_1: (T4, T1, T3, T2, T8)
P_2: (T5, T3, T1)
P_3: (T6, T8)
P_4: (T4, T2)

and listed in Table 2. Note that P_1 is the concatenation of test segments [T4, T1], [T1, T3], [T3, T2] and [T2, T8].

In step 5, the execution of Dijkstra's algorithm generates the shortest paths between pairs of nodes in the original FSM derived digraph G. They are listed below, in Table 3.

Table 3. The shortest paths between pairs of nodes in G

state	①	②	③
①	∅	(T_1)	(T_2)
②	(T_3)	∅	(T_3,T_2)
③	(T_5,T_3)	(T_5}	∅

In step 6, the auxiliary graph G_2 is constructed as below: there are four nodes, each corresponding to one segment sequence generated in step 4. The nodes are:

n_1: (T4, T1, T3, T2, T8)
n_2: (T5, T3, T1)
n_3: (T6, T8)
n_4: (T4, T2)

The end state of a segment sequence is the end state of its last transition, and the start state of a segment sequence is the start state of its first transition. Therefore the start/end state of each segment sequence can be obtained from Table 1. Based on Table 3, we now work out the edges (and their weights as well) between any pair of nodes in G_2 as below:

From n_1 to n_3: The end state of segment sequence n_1: (T4, T1, T3, T2, T8) is state ③ in G (i.e., the end state of T8). The start state of segment sequence n_3: (T6, T8) is state ① in G (i.e., the start state of T6). By definition, there is a directed edge $<n_1, n_3> \in E_2$, and the weight of the edge is the number of transitions in the shortest path from state ③ to state ① in the original FSM derived graph G. From Table 3, this shortest path is (T5, T3). Thus the weight of the edge is 2.

All other edge weights are calculated similarly, and are listed in the Table 4. For ease of understanding, not only the weights of edges but the shortest paths between nodes are also listed in the table.

Table 4. Edge weights between nodes of G_2 (*sp*: shortest path between nodes, *w*: weight)

	n_1		n_2		n_3		n_4	
	w	*sp*	*w*	*sp*	*w*	*sp*	*w*	*sp*
n_1			0	∅	2	(T5,T3)	1	(T5)
n_2	0	∅			2	(T3)	0	∅
n_3	1	(T5)	0	∅			1	(T5)
n_4	1	(T5)	0	∅	2	(T5,T3)		

For ease of tracking, the weight of each edge in Figure 5 is shown by a sequence of transitions (instead of the number of the transitions).

In step 7, a shortest tour of G_2 is generated using ATSP algorithm found in [7]. One of the shortest path is shown in thick dotted lines in Figure 5, i.e., ((T6, T8), (T5, T3, T1), (T4, T2), (T4, T1, T3, T2, T8)) with total traveling weight 1, which corresponds to only one transition, $T_{5,}$ as shown beside the edges of the paths in Figure 5.

The transitions on the edges of the shortest path of G_2 are those we need to link from one segment sequence to the next. In Step 8, the transitions inside the nodes and those on edges along the path are listed, and thus we get the following test sequence

T6,T8,T5,T3,T1,T4,T2,$\underline{\text{T5}}$,T4,T1,T3,T2,T8 where underlined transitions (e.g., T5 in this example) are those connecting consecutive segment sequences.

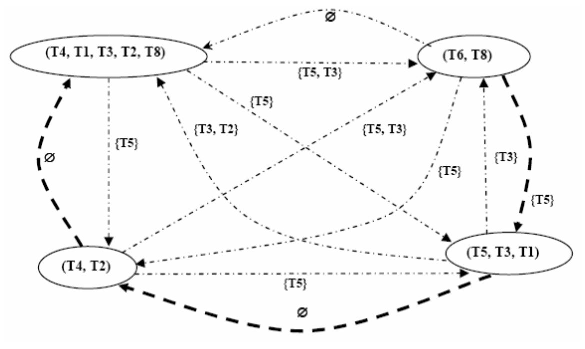

Figure 5. G_2 with one of its shortest path marked in thick dotted lines.

The shortest tour of a graph is, generally, not unique. For instance, a different shortest path is shown in thick dotted lines in Figure 6. This results in another final test sequence,

T6,T8,<u>T5</u>,T4,T1,T3,T2,T8,T5,T3,T1,T4,T2

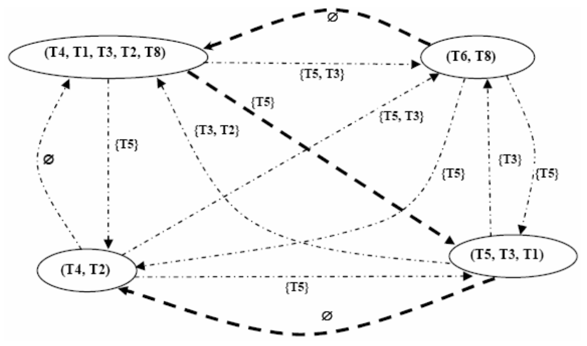

Figure 6. G_2 with another shortest path marked in thick dotted lines

In both Figure 5 and 6, only one extra transition is needed to connect segment sequences in the final test sequence. In fact, once G_2 is given, the number of transitions used for linking segment sequences of the final test sequence is fixed because the total weight of the shortest path covering every node of G_2 exactly once is fixed. Therefore, the total cost of connecting segment sequences is fixed. As there is no cost in connecting test segments within the segment sequences in phase 3, the total cost (i.e., *l*) of connecting test segments is minimized.

As a comparison, the total length of the final test sequence generated using our approach is 13 (as described above), while it is 23 using Chen's approach (see [3] for detailed test sequence generation).

5. DISCUSSION

In Section 4, we converted the problem of finding the shortest test sequence to finding a TSP path[7] over the auxiliary graph G_2. In this section, we discuss some issues regarding our test sequence generation approach.

Firstly, the TSP is NP-hard [12]. There exists a large number of heuristics in literatures for solving TSP. The TSP algorithms can be broadly divided into two categories: *symmetric* and *asymmetric*. The symmetric TSP is a special case of asymmetric TSP because the symmetric TSP heuristics deal with undirected graphs while asymmetric TSP algorithms deal with digraphs. We adopt asymmetric TSP in this paper.

Secondly, a TSP solution is a minimum-cost tour covering every node of the graph exactly once. One of the easiest ways to produce a TSP path from the TSP solution set is by breaking the TSP tour (say, from an edge of the highest weight). On the other hand, conformance testing requires that a test sequence start from the initial state s_0 of the FSM derived graph G. If s_0 is the start state of a segment sequence, once a TSP tour is produced, we can simply break the TSP tour such that the first edge (or transition) of the resultant TSP path starts from s_0. This guarantees that the generated test sequence starts from the initial state of the FSM M. If s_0 is not the start state of a segment sequence, we can select a segment sequence in the TSP tour such that the shortest path from s_0 to the start state of the segment sequence is minimum (recall that the shortest paths has been produced in Step 5 of algorithm TestSeqGen( )). We then break the TSP tour before that segment sequence of the TSP tour, and then add the shortest path from s_0 as the initial path to the test sequence. In any of these cases, the reset function is not required.

Theoretically, Phase 3 of our algorithm is optional. That is, without this phase, it is still possible to generate the final test sequence. However, phase 3 would reduce the number of nodes of G_2 in Phase 4. As TSP is NP-hard, any reduction in the number of nodes in G_2 would significantly improve performance of the algorithm.

Finally, we show that due to the assumption that the specification FSM and the implementation FSM are both minimal, the exclusion of some test segments in Phase 2 and the concatenation of segment sequences in Phase 3 do not affect the fault detecting capability.

In the Wp method, an error is detected if there are some unexpected outputs corresponding to inputs in a test segment. By finding which test segment the error is detected, we can determine the nature and the location of the error. Segment sequences in our method have the explicit correspondence relation to those segments in the Wp method. To show this, we only need to ensure that any unexpected output which is detected by a Wp method test segment will also be detected by the segment sequences generated using our method.

The proof is trivial for conformance testing. It is obvious that our test sequence can actually be viewed as a Wp test sequence for each of the test segment, with alternative prefix segment and

[7] A TSP path of a graph is a shortest path that goes through all nodes of the graph exactly once [22].

homing sequence. As the prefix segment and homing sequence in a test sequence do not influence the fault detection capability of a test segment in the test sequence, our test sequence has the same fault detecting capability as each of its embedded test segments. As a matter of fact, our test sequence can be viewed as a unified test sequence for all the generated Wp test segments.

6. CONCLUSION

This paper has described a graph-based technique that reformulates the problem of the test sequence generation based on the Wp method for a given FSM into one of finding the shortest path in an ATSP, thus enabling the use of one of the many existing meta-heuristic algorithms in conformance testing. We first defined the optimization objectives for generating the optimal test sequences. Then we presented an approach to generate shorter test sequences in a series of steps, each reducing the length of the final test sequence in some way. The approach started with exclusion of redundant test segments, followed by concatenation of test segments without linking cost. The problem of finding the shortest test sequences was finally converted to the well-known asymmetric TSP problem, of which we have include URL of online TSP sources and documentations. We have shown that our approach reduces the length of the test sequences required for conformance testing while maintaining the same fault detection capability.

7. REFERENCES

[1] Aho, A. V., Dahbura, A. T., Lee, D., and Uyar, M. U. An optimization technique for protocol conformance test generation based on UIO sequences and rural Chinese postman tours, *IEEE Transactions on Communications,* 39(11): 1604-1615, 1991.

[2] Chen J., Hierons, R. M., Ural, H., and Yenigun, H. Eliminating Redundant Tests in a Checking Sequence, Proceedings of TESTCOM 2005, LNCS 3502, 2005, pp. 146-158.

[3] Chen, W. H. An Optimization Technique for Protocol Conformance Testing Based on the Wp Method, *International Journal of Applied Science and Engineering*, 1(1): 2003, pp. 45-54.

[4] Chow, T. S., Testing Design Modelled by Finite-State Machines, *IEEE Transactions on Software Engineering* 4 (3), 1978.

[5] Fujiwara, S., Bochmann, G. V., Khendek, F., Amalou, M., and Ghedamsi, A. Test selection based on finite state models, *IEEE Transactions on Software Engineering*, 17: 1991, pp. 591-603,.

[6] Lee, D., and Yannakakis, M. Principles and methods of testing finite state machines, *Proceedings of IEEE,* 84 (8): 1996, 1090-1123.

[7] Online TSP source and documentation: http://www.furchur.de/de/informatik/laborberichte3.htm

[8] Prem, G. P., and Venkataram, P. Protocol Test Sequence Generation Using MUIOS Based on TSP Problem, *Proceedings of National Conference on Communication (NCC'93)*, Madras, Dec. 30- Jan. 2, 1994, pp.156-175,.

[9] Vijay Kumar, B. P., and Venkataram, P. An optimization technique for Protocol conformance test sequence generation based on MUIO using Hopfield Neural Network, *Proceedings of IFIP TC6 Conference* - pet.ece.iisc.ernet.in, 1994.

[10] Antoniol, G., Di Penta, M. and Harman, M. Search-Based Techniques Applied to Optimization of Project Planning for a Massive Maintenance Project, *Proceedings of the 21st IEEE International Conference on Software Maintenance (ICSM 2005)*, September 2005, Budapest, Hungary, pp. 240-249.

[11] Harman, M., Hu, L., Hierons, R. M., Wegener, J., Sthamer, H., Baresel, A. and Roper, M. Testability Transformation, *IEEE Transactions on Software Engineering*, Vol.30, No. 1, 2004, pp. 3-16.

[12] Lawler, E. L. *Combinatorial Optimization: Net-works and Matroids*. Holt, Rinehart and Winston, New York, 1976.

[13] Wikipedia The free encyclopaedia, http://en.wikipedia.org/wiki/Traveling_salesman_problem.