Fault Identification Using a Finite State Machine Model with Unreliable Partially Observed Data Sequences

Anastasios T. Bouloutas, George W. Hart, and Mischa Schwartz, *Life Fellow, IEEE*

Abstract—**In this paper we address the problem of minimum cost identification of a finite state machine using a trace of its event history. The motivation for this work is fault identification in communication systems. Other applications are possible as well. The event history we are using for the identification is partially observed, i.e., it is known to be a member of a regular language. Any string which belongs in this regular language is a possible trace of the FSM's event history. Furthermore, the event history is assumed to be corrupted with deletions, additions, and changes of symbols. The finite state machine to be estimated is related to a known finite state machine by performing an unknown number of additions and changes of arcs. An identification algorithm is developed based on a fast algorithm which can correct corrupted data strings generated by a known finite state machine. Examples of the method are provided, including one based on the IEEE 802.2 Logical Link Control protocol. This work is part of a continuing effort to develop a unified approach to fault management in communication networks.**

I. Introduction

THE process of fault identification in complex systems in general, and in large communication systems in particular, can be divided into the following three steps.

The first step is fault detection. Fault detection can be thought of as an on-line process with the purpose of giving an indication that some component is malfunctioning. Explicit or detailed identification of the malfunctioning component is not required at this stage.

The second step is the analysis of the data or output from the malfunctioning component in order to propose possible hypotheses of faults.

The third step is the fault localization process given a number of possible hypotheses of faults. Testing may be the most appropriate way of isolating the fault at this stage.

In this paper we concentrate on the second step of this process. Fault detection was discussed in an earlier paper [3]. We provide a framework for analyzing the data generated by a malfunctioning component of a communications network in order to propose possible hypotheses of faults. We envision an architecture in which the communication history of each node in a communication network is recorded. Once a fault appears, an off-line identification unit is notified and the history of the communication is used to propose possible hypotheses of faults.

One important task is to adopt an appropriate model for the behavior of communication systems. A generally accepted model is that of a discrete event system. A discrete event system is one whose behavior is adequately described by the sequences of discrete events it can execute. Many different discrete event formalisms have been proposed for modeling communication systems. Finite State Machines, Petri-nets, Calculus of Communicating Systems, and Communicating Sequential Processes are some of the most important. In this work we concentrate on Finite State Machine models. Finite state machine models have been used extensively to specify the behavior of single physical systems or specific protocol units. This work concentrates on fault identification of a single protocol unit, or a single system represented as a finite state machine. This provides a unifying principle for handling fault identification in a variety of systems or subsystems.

Fault localization methods are assumed to have isolated a fault in a single process of a communication system. Often, fault isolation in a communication process is not enough. Possible hypotheses about the nature of the fault are very important in the process of identifying and correcting the fault. In this work we assume that the fault has been isolated to a single process, but the nature of the fault is unknown, i.e., we do not know what the problem is specifically. The communication process is represented as a finite state machine, and the possible faults are represented as additions and changes of arcs in that FSM. We use the recorded communication history (messages exchanged by the communication process) in order to propose possible hypotheses of multiple faults. Furthermore, we assume that the recorded communication history is partially observed (i.e., we may not have the capability to observe all the events), and possibly corrupted (i.e., the observed communication history may include deletions, additions, and changes of events). Thus, the problem becomes one of inferring a finite state structure given unreliable, partially observed information about its event trace history. In this paper we examine the above-defined problem in a general way which may be useful to a variety of applications. However, our motivation is fault

Paper approved by K. Sabnani the Editor for Communication Protocols of the IEEE Communications Society. Manuscript received May 14, 1990; revised November 7, 1991. This work was supported by NSF under Grant CDR 88-11111 by EPRI under Grant 8000-32, and by CAT under Grant CU01125801-006.

A. T. Bouloutas was with the Department of Electrical Engineering, Columbia University, New York, NY. He is now with the IBM Corp., T. J. Watson Research Center, Yorktown Heights, NY

G. W. Hart and M. Schwartz are with the Department of Electrical Engineering and Center of Telecommunications Research, Columbia University, New York, NY 10027.

IEEE Log Number 9211130.

identification in communication networks. In Section VII we give an example of how the method proposed here can be applied to fault identification in communication protocols.

The problem is to estimate a finite state structure given unreliable, partially observed information about its event trace history. We assume that the finite state structure to be estimated can be constructed by changing a known finite state structure in specific ways. The class of changes considered here consists of replacement of arcs by others, and addition of arcs. Addition and deletion of states, and deletion of arcs are not considered, but the proposed algorithm can be extended to include these changes as well.

We assume that the event trace history we are using for the estimation is part of an execution sequence of the unknown machine, which has been corrupted with "noise." Furthermore, only a partial observation of the corrupted sequence is assumed. Partial observation means we simply know that the corrupted sequence is a member of a regular language, i.e., any string which belongs in the regular language is a possible event trace history representing a corrupted data sequence. The class of errors considered in the event trace history is deletion of events, addition of events, and change of events.

This problem is a generalization of the well-known problem of inferring a finite state machine from its event trace history. In the past, a number of learning protocols for finite state structures have been considered. An excellent survey of the general area of inductive inference is presented in [1]. Most of the algorithms discussed in the literature assume that there exists a reliable event trace history which can be used in the identification process. Identification of finite state machines given unreliable observation of its event trace history had been considered by Hart [7], but he did not consider the case of limited observation on the data.

II. Definitions

An FSM is the 5-tuple

$$M = (Q, \Sigma, \delta, Q_0, F)$$

where Q is the finite set of states, $Q_0 \subseteq Q$ is the set of initial states, Σ is the set of events, or the alphabet, $F \subseteq Q$ is a set of final states, and $\delta \subseteq Q \times Q \times \Sigma$ is the transition relation. If $(q_1, q_2, \sigma) \in \delta$, by this means that the transition σ from state q_1 to state q_2 is possible. The symbol $\epsilon \in \Sigma$ denotes the unobservable transition.

A trace or sequence s is the concatenation of a finite sequence $s = [\sigma_1, \cdots, \sigma_n]$ of n elements of Σ, where [ ] denotes the empty sequence. Σ^* denotes the set of all strings $s = \sigma_0 \cdots \sigma_n$ such that $\sigma_0 \cdots \sigma_n$ are elements of Σ. A path p in M is defined to be $[(q_1, \sigma_1), (q_2, \sigma_2), \cdots, (q_{n+1}, \emptyset)]$ such that $(q_i, q_{i+1}, \sigma_i) \in \delta, i = 1, 2, \cdots, n$, with $q_1 \in Q_0$ and $q_{n+1} \in F$. A word generated by M is defined to be any trace $s = [\sigma_1, \sigma_2, \cdots, \sigma_n]$ such that there exists a path p in M, with the same transitions $\sigma_1, \cdots, \sigma_n$ and in the same order as in s. The set of all the words that can be generated by M is defined to be the language of $M, L(M)$.

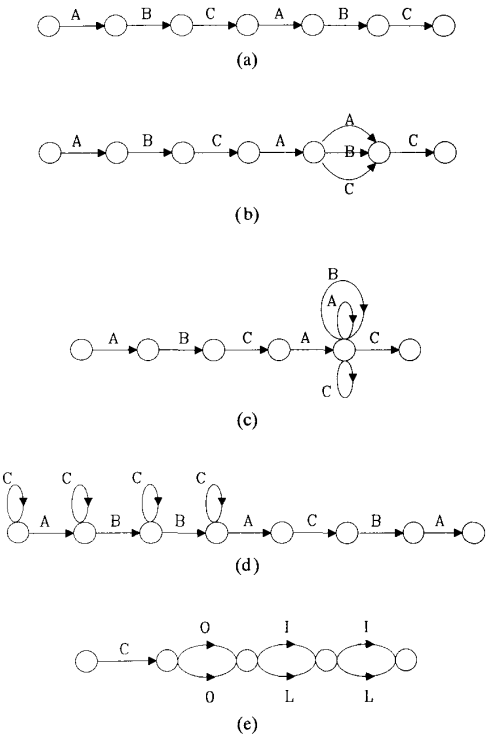


Fig. 1. Applications of the representation of the data as an FSM.

III. Model of the History

In our model we assume that the available event trace history is a partial observation of the event trace history which has been generated by the finite state structure to be estimated. Since the event traces are partially observed, there may exist many sequences of events compatible with a given observation. The set of possible event traces is represented by a finite state machine called the data, or history, finite state machine. Any trace in the finite state machine representing the data could be an event trace of the unknown structure. This broad notion of history is introduced because it can describe a wide range of typical data corruptions and yet is amenable to efficient data reconstruction algorithms [2], [11].

The simplest and most important type of history is when the observation is complete, so the data FSM consists of a "single chain" of states as in Fig. 1(a).

A typical problem where our broader formulation is valuable is when a "burst of noise" causes a particular symbol to be unrecognizable. The FSM of Fig. 1(b) shows how the sequence $A \cdot B \cdot C \cdot A \cdot ? \cdot C$ could be represented. If the "burst of noise" corrupted a sequence of unknown length, then the FSM of Fig. 1(c) would be used.

Another type of limited observation can be modeled with the same formalism. Assume that we receive strings in $\{A, B, C\}^*$[1], but for some reason we could not observe C during the first half of the data collection period. Then the

[1] Here * denotes the Kleene closure.

sequence

$$\underbrace{A \cdot B \cdot B \cdot A}_{C \text{ channel not operating}} \cdot A \cdot C \cdot B \cdot A$$

can be modeled as shown in Fig. 1(d). Here an unspecified number of C's can be interleaved into the first half of the observed data sequence.

The range of applications of this kind of modeling is quite large. This model is applicable in many situations in which we want to reconstruct the data generated by a discrete system, but we do not have full observation of the event trace history of the system. Our motivation for employing this model is fault finding in communication processes, where the identification unit cannot observe all the events that take place. Another example application could be in text processing. Assume that we have an optical reader which confuses symbols "I" and "L"; it also confuses symbols "O" and "0". The string of text

$$C \cdot O \cdot I \cdot I$$

can be modeled as shown in Fig. 1(e).

IV. The Model of the System

As shown in Fig. 2, a known finite state machine, θ_c (correct structure) has changed (perhaps due to a failure) to a new finite state structure θ to be estimated. Events generated by the unknown structure pass through a corruption process where symbols are added, deleted, and changed. The corrupted data, or event sequence h, is partially observed, and is represented by the data FSM D, which is presented to an estimator. The estimator has to generate the best estimate for the changed structure θ, as well as the uncorrupted event history h_c (correct history) generated by structure θ.

The estimator can estimate h_c by choosing one particular path in D and changing, inserting, and deleting symbols to correct for the data corruption process. The estimator can alter the model of the structure by changing and adding arcs.

Each of the operations, deletion of a symbol, addition of a symbol, and change of a symbol in the event history has been assigned a cost. Each of the operations of changing an arc, and adding an arc in the structure, has been assigned a cost as well. All of these costs must be chosen in advance. Different choices may give rise to different estimators, as is always the case with estimation.

The purpose of this work is to propose a design for the estimator of the minimum-cost change criterion. Under this criterion, the sum of the cost of the estimated event history and the cost of the estimated structure should be minimum, under the constraint that the estimated event history is a possible word of the estimated finite state structure.

Let h be a sequence generated by the data finite state machine D, i.e., a possible event history. The length of sequence h is N. The set of operations the estimator can perform on sequence h are deletions, additions, and changes of symbols. The operation of deleting symbol x in the nth position of event history h is defined to be d_x^n, and the cost associated with this operation is $C(d_x^n)$. The operation of inserting symbol x in the nth position of event history h, i.e., after the $n-1$st symbol is i_x^n, and the cost associated with this operation is $C(i_x^n)$. The operation of changing symbol x to symbol y in the nth position of data sequence h is c_{xy}^n, and the cost associated with this operation is $C(c_{xy}^n)$.

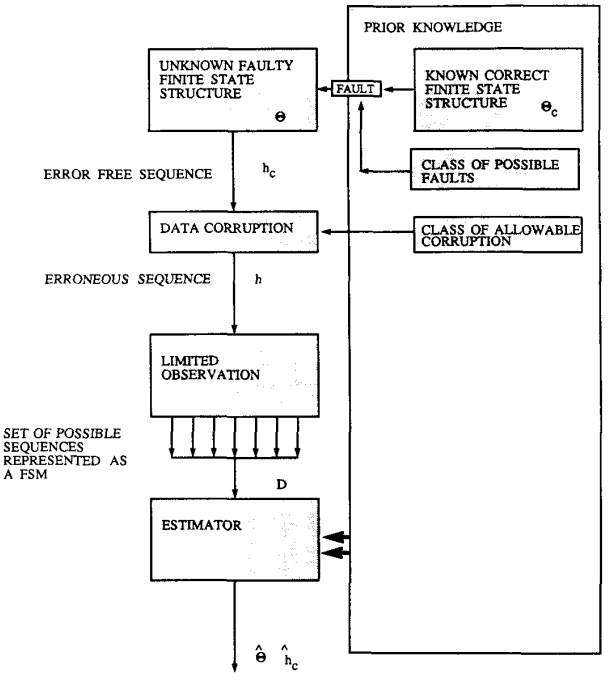


Fig. 2. Model of the system.

We should stress here that the insertions, deletions, and changes of symbols refer to operations performed by the estimator. The operations performed in the sequence which is generated by the unknown structure, in order to get the corrupted sequence, are symmetrical. Thus, if symbol x is deleted in the corruption process, the corresponding operation of the estimator (in order to restore the sequence) should be i_x, i.e., to insert symbol x.

The sequence representing the operations on the corrupted event history h is defined to be f_{h_c}. The elements of f_{h_c} belong to the set $A = \{d_x^n, c_{xy}^n, i_x^n \mid x, y \in \Sigma, 1 \le n \le N\}$. They operate from left to right. For example, the sequence of operations $f_{h_c} = c_{AB}^1 \cdot c_{BC}^2 \cdot d_C^3 \cdot i_A^3$ if applied to the sequence of observed data, $h = A \cdot B \cdot C$, gives the sequence $h_c = f_{h_c}(h) = B \cdot C \cdot A$. Note that a deletion of a symbol followed by an insertion of another symbol is equivalent to changing one symbol to a new one. The operation which will be chosen is the one with the least cost.

The cost associated with the sequence representing the operations, i.e., sequence f_{h_c}, is $C(f_{h_c}) = \sum_{i=1}^{|f_{h_c}|} C(z_i)$, where $|f_{h_c}|$ is the length of the sequence f_{h_c} and z_i is the operation $\left(c_{xy}^n, d_x^n, \text{ or } i_x^n\right)$ in the ith position of sequence f_{h_c}. Thus, the cost of sequence of operations is the sum of all the costs of additions, deletions, and changes of the labels that comprise the operation sequence.

In a similar fashion, one can define the sequence f_θ. The sequence f_θ is the sequence of operations on the correct finite

state structure θ_c. There are two classes of operations considered: addition of arc A in structure θ_c, a_A; and replacement of an arc A and B in structure θ, r_{AB}. The faulty structure is given by $\theta = f_\theta(\theta_c)$. The costs associated with operations a_A and r_{AB} are $C(a_A)$, and $C(r_{AB})$, respectively. The cost of the sequence f_θ is the sum of the costs of the operations that comprise this sequence, i.e., $C(f_\theta) = \sum_{i=1}^{|f_\theta|} C(z_i)$, where $|f_\theta|$ is the length of the sequence f_θ, and z_i represents the operation, a_A, or r_{AB}, in the ith position of sequence f_θ.

The output of the estimator is the minimum cost operation sequence $\hat{f}_{h_c,\theta}$, which consists of the estimated operation sequence on the structure and the estimated operation sequence on the data. This can be further analyzed in the operation sequence on the structure, $\hat{f}_\theta$, and the operation sequence on the data given the structure, $\hat{f}_{h_c|\theta}$, i.e., $\hat{f}_{h_c,\theta} = \hat{f}_\theta \cdot \hat{f}_{h_c|\theta}$. Here "$\cdot$" denotes the concatenation of two sequences.

The optimization is over all sequences h which can be generated by the data FSM D, and over all structures θ. Thus, we have to solve the following problem:

$$\begin{aligned}\hat{f}_{h_c,\theta} &= \arg_f \min\{C(f_{h_c,\theta})\} \\ &= \arg_f \min\{C(f_\theta) + C(f_{h_c|\theta})\}\end{aligned}$$

Such that

$$h \in L(D), \quad \text{and} \quad \hat{h}_c = \hat{f}_{h_c|\theta}(h) \in L\left(\hat{\theta}\right), \quad \text{with} \quad \hat{\theta} = \hat{f}_\theta(\theta_c).$$

Here the term $\arg_x \min f(x)$ gives the argument x which minimizes function $f(x)$.

This is a hard optimization problem. In order to tackle it, we use local search in the space of possible structures θ. This method may lead to suboptimal solution, but since the problem of identifying a finite state machine is NP-Complete [5], only heuristic solutions can be realistically considered.

V. The Algorithm

The proposed algorithm relies on an earlier fast algorithm, a modification of the well-known Viterbi algorithm, which identifies the optimum changes in the data, given the finite state structure [2], [11]. The proposed algorithm starts by searching through the space of the possible operations on the structure θ_c, f_θ. Given an f_θ, the algorithm finds $\theta = f_\theta(\theta_c)$ and associates it with the cost $C(f_{h_c,\theta})$. This is, as noted earlier, the sum of the costs $C(f_\theta)$ and $C(f_{h_c|\theta})$. The cost $C(f_\theta)$ is easily determined from the cost tables, and the algorithm presented in [2] finds $C(f_{h_c|\theta})$. Thus, each new structure is associated with cost $C(f_{h_c,\theta})$. The algorithm continues the search until the number of operations in f_θ reaches a certain prespecified bound N, i.e., $|f_\theta| = N$. The algorithm outputs the structure with the lowest cost seen in the search until this point.

This algorithm can be formally presented as follows.

Input:

a finite state machine θ_c, which represents the given structure;

a finite state machine D, which represents a set of possibly unreliable observed data histories;

a table which gives the cost of the different operations, $c_{xy}^n, d_x^n, i_x^n, r_{AB}, a_A$.

Output:

The operation sequence $\hat{f}_{h_c,\theta}$ with the minimum cost, the corresponding structure $\hat{\theta}$, and the data history h_c.

We now present the method. Let L_n be the list which contains the k best structures which can be generated with n operations from structure θ_c. Here k is a parameter of the algorithm which can be changed to tune its operation. Let D represent the given finite state machine which describes the set of possible faulty histories, θ_c the given structure, f a set of possible operations $\left(c_{xy}^n, d_x^n, i_x^n, r_{AB}, a_A\right), C(f_{h_c,\theta})$ the cost associated with the operations on the history and the structure, λ the null operation or no operation at all, and N the maximum number of operations we would like to consider. *Best_structure* is a variable which holds the triplet (*operations, structure, cost*) with the lowest cost generated by the algorithm so far. *Best_k_structures*(L) is a function which, given L, a list of structures with more than k elements, simply extracts as output a list which consists of only the k structures with the lowest cost. *Best* (L) is a function which, given L, a list of structures, extracts as output the one with the lowest cost.

In the initial step, the algorithm assigns to the list L_0 the null fault, the correct structure, and the cost associated with the correct structure. The cost associated with the correct structure is the one corresponding to the operations of the history such that this history is accepted by the correct structure, as given by the algorithm in [2].

Method:

Step 1: $f \leftarrow \lambda; n \leftarrow 0; L_0 \leftarrow (\lambda, \theta_c, C(f_{h_c,\theta_c}))$; *Best structure* $\leftarrow L_0$.
(*Comment:* In Steps 2–5, we construct the list L_{n+1} by operating on the elements of list L_n.)

Step 2: Get the first element $(f, \theta, C(f_{h_c,\theta}))$ of L_n; $L_n \leftarrow L_n - \{(f, \theta, C(f_{h_c,\theta}))\}$.

Step 3: Let $(f, \theta, C(f_{h_c,\theta}))$ be the element of L_n selected in Step 2. θ has previously been generated by applying n operations on θ_c. In this step we generate a list *Succ*. The elements of list *Succ* are all the structures θ' that can be generated by applying a single additional operation on structure θ. Thus, if $\theta' \in$ *Succ*, then θ' can be generated by applying $n+1$ operations on θ_c, i.e., *Succ* is the list of successors of θ, in which each element has the form (*operation, structure, cost*) in which the operation is extended by one addition or replacement, the structure is accordingly modified, and the cost has been increased according to the table entry of the operation. The details of our method of choosing the operation are described below in Section VI.

Step 4: $L_{n+1} \leftarrow$ *Best_k_structures*$\{L_{n+1} \cup$ *Succ*$\}$; *Best_structure* $\leftarrow$ *Best* {*Best_structure, Best* (*Succ*)}.

Step 5: Is L_n empty? If NO, go to Step 2. If YES, $n \leftarrow n+1; L_n \leftarrow L_{n+1}$.

Step 6: If $n < N$, go to Step 2. Otherwise, output *Best_structure* and stop.

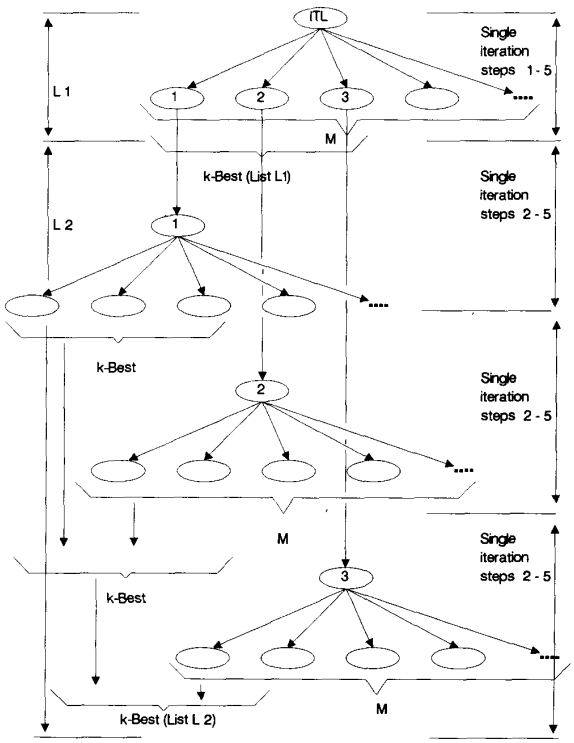


Fig. 3. Two initial steps of the algorithm. Ovals represent triplets of operations, structure, and cost.

The execution of the first $k+1$ iterations of this algorithm with $k=3$ and the generation of lists L_0, L_1, L_2 are presented in Fig. 3. Assume a given structure generates M possible successor structures, i.e., the maximum number of elements of list *Succ* is M. Out of the M successor structures, we choose k for list L_1. Since L_0 contains a single element, L_1 is generated by a single execution of Steps 1–6. After this execution, L_1 contains k elements. This concludes the first iteration of the algorithm. In the next k iterations (execution of Steps 2–5 above k times), each of the k elements of L_1 is expanded (one at each iteration). Finally, out of the possible Mk elements (each of the k elements has M successors), k are chosen for the list L_2

As can be seen, this algorithm is a pruned breadth-first search in the set of possible structures. The algorithm executes Steps 2–5 kN times. Here N is the maximum number of operations we are considering.

The algorithm generates lists $L_1, \cdots, L_N$. We explain here one step, i.e., how it proceeds to generate L_{n+1} from L_n. Assume that the algorithm has identified k structures in list L_n. Each of these structures is n operations different from the original structures, i.e., for each structure in L_n, $|f_\theta| = n$. The algorithm has to identify k structures by executing Steps 2–5 k times. Each of these structures (i.e., the elements of L_{n+1}) has $n+1$ operations different from the original structure.

The algorithm expands one by one the elements of L_n, by performing one more operation on them. At each iteration of Steps 2–5, one element is expanded. Assume that each element has at most M successors. The successors of each element of L_n are inserted in list *Succ*. Effectively, the algorithm expands all elements of L_n and chooses the k best (with the lowest cost) which are inserted in list L_{n+1}. It performs the previous task in k iterations of Steps 2–5, in order to use less memory. In Step 4, the algorithm combines the elements of L_{n+1} with the M elements of list *Succ*. In general, list $L_{n+1} \subset$ *Succ* has $k+M$ elements, although in the first iteration, i.e., when $n=1$, list L_1 is empty. Thus, $L_1 \subset$ *Succ* has M elements. Among the $k+M$ elements, the k elements with the least combined cost are chosen. The algorithm proceeds in this way and generates list L_{n+1} expanding, in each iteration, one of the elements of list L_n. The memory required for this scheme is $O(k+M)$, certainly less than $O(kM)$ one would need if all the elements of L_n were expanded in one iteration.

It is shown in [4] that the complexity of this algorithm is $O\left((|\theta||D|)^2 NMk\right)$, where $|\theta|$ and $|D|$ denote the number of states in FSM's θ and D, respectively.

VI. Generation of New Structures

We now describe Step 3 of the algorithm, i.e., the generation of new structures. The algorithm given in Section V is very general. It need not be a data-driven approach. One can introduce in list *Succ* the structures with the most common operations, or use some other criterion which does not depend on the given data. We choose, however, to implement a data-driven approach. Given a structure θ which belongs in the list L_n, the algorithm generates all structures that can be generated by θ with one operation such that the new structure accepts the data "a little better." All these structures are inserted in list *Succ*.

If there exists an event trace history $h \in L(D)$ which is accepted by the given structure θ, then *Succ* is empty and the algorithm proceeds to Step 5. If there does not exist an event trace history $h \in L(D)$ which is accepted by structure θ, then the algorithm makes a set of hypotheses of structures which accept the history "a little better."

The algorithm follows in parallel all the paths in the data that are accepted by the structure as far as possible until the data cannot be accepted by the FSM, or one of the final states is reached. If a point is reached where a particular path in the data is not accepted by the structure, the algorithm inserts in *Succ* all the structures θ' which can be generated from θ using a single operation and which can accept the next label of this data path.

A detailed description of the algorithm to generate new structure can be found in [4]. Summarizing, this algorithm gets as input a finite state structure θ and a set of data D represented as a finite state machine. It gives as output a set of structures θ' which differ from the given structure θ by a single operation and accept the data "a little better."

VII. Examples

The algorithm presented in Section V has been implemented in LISP, and has been tested in various examples of structures

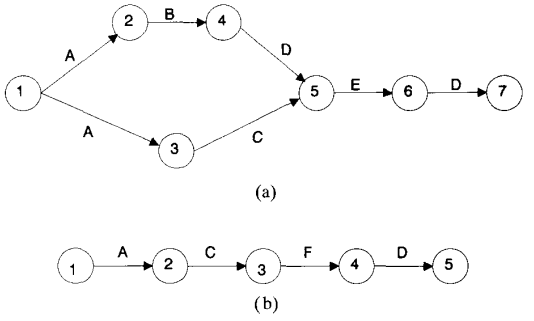


Fig. 4. Example one. (a) Representation of the structure S; (b) representation of the data D.

and data. We present here four of these examples—three of a simple tutorial nature, and one showing how this theory can be applied to a complex communications process. The first is the example presented in Fig. 4. The maximum number of operations we consider is 4, and k, the maximum number of structures we keep in each iteration of the algorithm, is chosen to be 4. The costs associated with each of the operations are:

Operation	Cost
change a symbol in data	1
insert a symbol in data	1
delete a symbol in data	2
replace an arc in structure	2
add an arc in structure	3

The results is that the algorithm changes the label F in the data to E, i.e., it changes the transition (3, 4, F) to (3, 4, E) at cost 1.

The same example was tested with different cost functions. In a second run, the costs were as follows:

Operation	Cost
change a symbol in data	2
insert a symbol in data	1
delete a symbol in data	2
replace an arc in structure	1
add an arc in structure	3

The result was to replace the arc (5, 6, E) in the structure with (5, 6, F) at cost 1. In both cases, it can be verified that the given solutions are the minimum cost ones.

The second example is the one presented in Fig. 5. The costs assigned here are again:

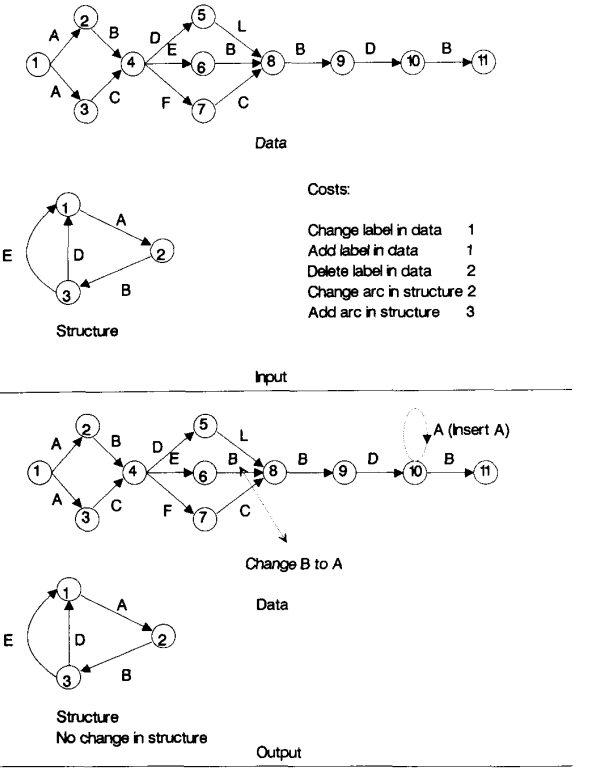


Fig. 5. Example two.

Operation	Cost
change a symbol in data	1
insert a symbol in data	1
delete a symbol in data	2
replace an arc in structure	2
add an arc in structure	3

The algorithm identified two changes in data. The transition (6, 8, B) in the data changes to (6, 8, A) and a new transition is added in data, namely, transition (10, 10, A) which, at cost 2, is the minimum cost solution.

A third example considered is the one presented in Fig. 6. The following costs were introduced:

Operation	Cost
change a symbol in data	2
insert a symbol in data	1
delete a symbol in data	1
replace an arc in structure	1
add an arc in structure	3

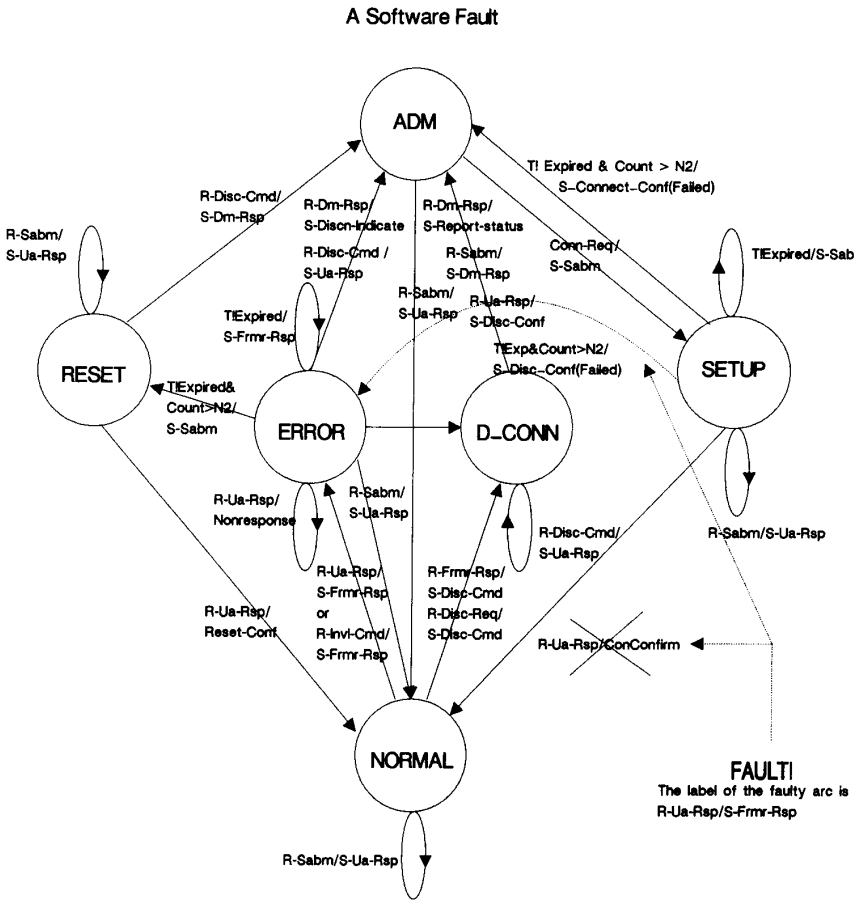


Fig. 7. Part of the IEEE 802.2 logical link control protocol.

The algorithm generates one change in the structure, arc (1, 2, A) replaced by $(1, 2, B)$, and one change in the data, arc $(1, 2, A)$ changed $(1, 2, B)$, which can be verified to be the minimum cost solution.

As a last example, we introduce and discuss how we can use this algorithm to identify software faults in protocols. Software faults may be presumed to have different characteristics than hardware faults. They may likely be due to a human error in implementing the state diagram of a protocol unit in software. Improper testing and verification allow them to remain undetected. Some of the probable faults that may happen are: deletion of a transition, addition of a transition, and change of a transition in the state machine that describes the protocol operation.

The example chosen was a software fault in the IEEE Logical Link Control 802.2 protocol. A portion of the state machine appears in Fig. 7. We designed the following possible scenario of a software fault: assume that we have two 802.2 protocol units implemented in software that communicate with each other. One of them is assumed to have the fault shown in Fig. 7, i.e., from state *SETUP*, the faulty unit goes to state *ERROR* if it receives *UA_RSP*. It transmits an *FRMR_RSP* in response.

Transition *UA_RSP/CONN_CONF* connecting state *SETUP* to *NORMAL* is not assigned. This fault might happen if the programmer (by mistake) assigned the label *UA_RSP/FRMR_RSP* from state *SETUP* to state *ERROR*, instead of assigning the label *UA_RSP/CONN_CONF* from state *SETUP* to state *NORMAL*.

This kind of error can go unnoticed for a long period of time since it does not affect the operation of the protocol if the faulty unit does not initiate the communication. However, a problem arises when the faulty unit initiates the process to establish communication. Fig. 8 shows a possible communication history where this problem becomes apparent. We assume that the problem exists in unit B.

The signals *SABM* are sent by both units at approximately the same time. Both units then enter the *SETUP* state. Each unit, upon receiving the *SABM* that was sent by the other site, stays in *SETUP* state and transmits *UA_RSP*. Upon receiving *UA_RSP*, site A enters state *NORMAL*, while site B, which has the problem, enters the *ERROR* state transmitting *FRMR_RSP*.

Upon receiving the *FRMR_RSP*, site A goes from the *NORMAL* state to the state *D_CONN* and sends a *DISC_CMD*. When the *DISC_CMD* is received, site B goes from state

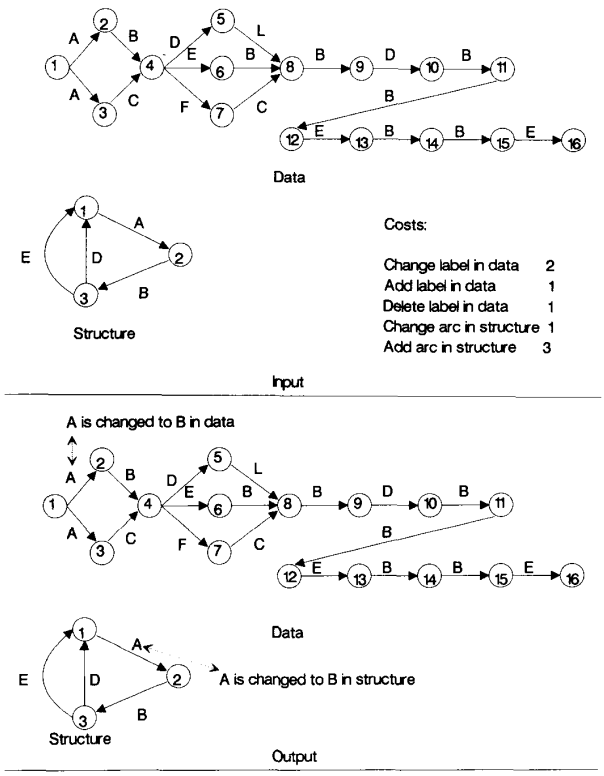


Fig. 6. Example three.

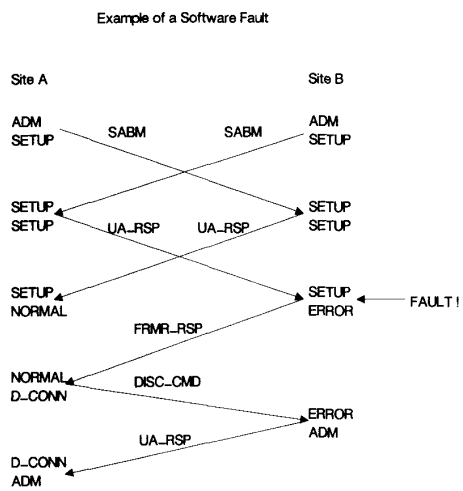


Fig. 8. An example of a software fault and the messages exchanged between the two sites.

ERROR to state *ADM* while sending a *UA_RSP* which is received by site A. *UA_RSP* makes site A change from state *D_CONN* to state *ADM*. Since both sites return to the state *ADM*, it is clear the attempt to initiate communication has failed. A monitoring unit would trigger the mechanism to check why this attempt to communicate was unsuccessful.

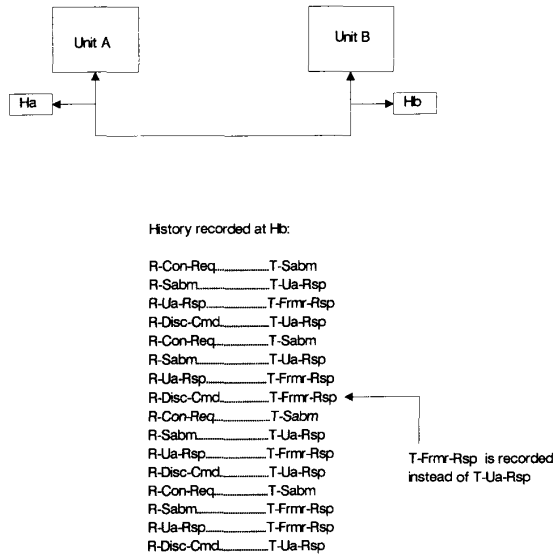


Fig. 9. Two history units and the history recorded at H_b.

To analyze this fault, consider the architecture of Fig. 9. Two history units, H_a and H_b, and the history recorded by unit H_b are shown there.

The off-line system identification unit accepts as input the description of the behavior of site A, the history h_A, the description of the behavior of site B, and the history h_B. It performs two different analyses—one for site A, using the model of A and h_A, and one for the site B and the history h_B.

The analysis of site A will not reveal any fault, while the analysis of site B should detect the possible fault. We focus here on the analysis of site B.

The history that has been recorded by the unit H_b is the following:

CON_REQ/SABM
SABM/UA_RSP
UA_RSP/FRMR_RSP
DISC_COMD/UA_RSP
CON_REQ/SABM
SABM/UA_RSP
UA_RSP/FRMR_RSP
DISC_CMD/FRMR_RSP (instead of DISC_CMD/UA_RSP)
CON_REQ/SABM
SABM/UA_RSP
UA_RSP/UA_RSP
UA_RSP/FRMR_RSP
...

Notice that at a certain point, the communication history is not recorded correctly. Due to channel noise or due to a random error, unit H_b has recorded *FRMR R_RSP* instead of *UA_RSP*, as shown in Fig. 9.

The model of the structure is the finite state machine that describes the correct behavior of the protocol unit. For this example, we assume that the possible faults are: replacement of an arc with some other arc, and an addition of an arc.

Deletion is again not considered since, in order to decide to delete an arc, we have to make some probabilistic decision (i.e., we have to decide that that arc is never used). The costs associated with these faults are:

Operation	Cost
change a symbol in data	1
insert a symbol in data	1
delete a symbol in data	1
replace an arc in structure	1
add an arc in structure	3

Note that the cost associated with noise (insertion, deletion, or change of a symbol) in the recorded history is 1, i.e., a packet error has the same weight, or the same probability, with a change of an arc due to human error. The model of the correct behavior of the protocol unit and the faulty recorded history are used as an input in the algorithm described in Section V. The result is that both faults—the one in the finite state structure (addition of the arc) and the one in the recorded history—are successfully recognized.

The implementation of this algorithm in LISP, on a SUN 3-50, in the case of the last example takes about 30 min to converge. A more careful implementation in C would make this method a viable alternative to testing the whole protocol unit.

The algorithm presented in this paper is sensitive to the cost assignments. Since the algorithm identifies the structures with the minimum cost change from a known structure, significant change in the costs will change the result. One way to avoid the problems that may be associated with incorrect assignment of costs would be to generate the k-best structures, instead of generating only the best structure. This requires a simple modification of the algorithm. In this case, the algorithm would generate several hypotheses which could be verified by testing. A detailed investigation of the sensitivity of this algorithm to incorrect assignment of costs is an issue for further research.

VIII. Conclusions

A method has been presented for the minimum cost identification of a finite state machine using partially observed possibly corrupted set of data. The method starts with a "seed" finite state machine and builds the machine to be identified using additions and changes of arcs. This method depends on a fast subroutine, presented in [2], to identify insertions, deletions, and changes in the data, given a hypothesized structure. The formulation is quite general and can be easily extended to the case of deletion of arcs, which is a topic for further research. A number of examples of a LISP implementation of this algorithm were given. One of the examples focuses on the identification of a possible fault in an implementation of the IEEE 802.2 protocol. The fault is identified using an unreliable trace of the communication history of the faulty protocol unit.

Work is continuing in this area in an attempt to unite various approaches to fault management in communication networks [8]–[11].

References

[1] D. Angluin and C. H. Smith, "Inductive inference: Theory and methods," *Computing Surveys,* vol. 15, no. 3, Sept. 1983.

[2] A. Bouloutas, G. W. Hart, and M. Schwartz, "Two extensions of the Viterbi algorithm," *IEEE Trans. Inform. Theory,* vol. 37, pp. 430–436, Mar. 1991.

[3] ——, "On the design of observers for fault detection in communication networks," in *Network Management and Control,* A. Kershenbaum, M. Malek, and M. Wall, Eds. New York: Plenum, 1990, Ch. 5.

[4] A. Bouloutas, "Modeling fault management in communication networks," Ph.D. dissertation, Columbia Univ., New York, NY, 1990.

[5] M. Gold, "System identification via state characterization," *Automatica,* vol. 8, pp. 621–636, 1972.

[6] ——, "Complexity of automaton identification from given data," *Inform. Contr.,* vol. 37, pp. 302–320, 1978.

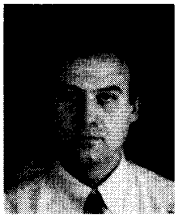
[7] G. W. Hart, "Minimum information estimation of structure," Ph.D. dissertation, MIT Dep. EECS, and Lab. Inform. Decision Sys. Tech. Rep. 1664, 1987.

[8] A. Bouloutas, G. W. Hart, and M. Schwartz, "Simple finite-state fault detectors for communication networks," *IEEE Trans. Commun.,* vol. 40, pp. 477–479, Mar. 1992.

[9] ——, "Identifying faults in communicating finite-state machines," submitted for publication.

[10] C. Wang and M. Schwartz, "Fault detection with multiple observers," *IEEE Trans./ACM Networking,* vol. 1, pp. 48–57, Feb. 1993.

[11] G. W. Hart and A. Bouloutas, "Correcting dependent errors in sequences generated by finite-state processes," to appear in *IEEE Inform. Theory,* July 1993.



Anastasios Bouloutas was born in Lamia, Greece, in 1959. He attended the National Technical University of Athens (NTUA), Greek, from 1977 to 1982. In 1983 he earned the Fulbright Fellowship to continue his studies at Columbia University, New York, NY. He received the M.Sc. degree in 1985 and the Ph.D. degree in 1989, both in electrical engineering.

From 1982 to 1983 he worked at the National Research Foundation of Greece. In January of 1990 he joined IBM T. J. Watson Research Center, Hawthorne, NY as a Post-doctoral Researcher, working on algorithms and prototypes for network management. From May 1991 to August 1992 he was with the National Technical University of Athens as a Research Scientist. Since September 1992, he has been with the IBM T. J. Watson Research Center.

George W. Hart received the B.S. degree in mathematics from the Massachusetts Institute of Technology in 1977, and the M.A. degree in formal linguistics from Indiana University in 1979. After four years as a Research Scientist at the M.I.T. Lincoln Laboratory and Energy Laboratory, he received the Ph.D. degree in electrical engineering and computer science from M.I.T. in 1987.

He joined the Faculty of the Department of Electrical Engineering, Columbia University, where he is currently an Associate Professor.

Mischa Schwartz (S'46–A'49–M'54–SM'54–F'66–LF'92) received the B.E.E. degree from Cooper Union, New York, NY, in 1947, the M.E.E. degree from the Polytechnic Institute of Brooklyn, NY, in 1949, and the Ph.D. degree in applied physics from Harvard University, Cambridge, MA, in 1951.

From 1947 to 1952 he was a Project Engineer with the Sperry Gyroscope Company, working in the fields of statistical communication theory, radar detection, and radar design. From 1952 to 1974 he was Professor of Electrical Engineering at the Polytechnic Institute of Brooklyn, serving as Head of the Electrical Engineering Department from 1961 to 1965. During the year 1965–1966 he was an NSF Science Faculty Fellow at the Laboratoire de Physique, Ecole Normale Superieure, Paris, France. During the academic year 1973–1974 he was a Visiting Professor at Columbia University, New York, NY. He joined that institution in September 1974 as Professor of Electrical Engineering and Computer Sciences. He is currently Charles Batchelor Professor of Electrical Engineering. For the 1980 calendar year, he was on leave as a Visiting Scientist with IBM Research. During 1986 he was on leave as a Resident Consultant with NYNEX Science and Technology.

Dr. Schwartz is a former Director of IEEE, formerly Chairman of the Information Theory Group, and past President of the Communications Society. He is author or coauthor of eight books on communications, signal processing, and computer communication networks, and has published extensively in the technical literature. He is on the editorial boards of *Networks* and *Computer Networks*. He has lectured extensively both in this country and abroad on various aspects of communication theory, communication systems, digital communications, and computer communications. He was a recipient of a Distinguished Vistor Award in 1975 from the Australian-American Education Foundation. He is a Fellow of the AAAS, and past Chairman, Commission C., U.S. National Committee/URSI. He has been a consultant on communications, computer communications, signal processing and radar to many companies. In 1983 he was the recipient of the IEEE's annual Education Medal. Columbia University awarded him its Great Teacher medal in 1983. In 1984 the IEEE Centennial year, he was cited as one of the 10 all-time electrical engineering educators. He served from 1985 to 1988 as Director of the Columbia University Center for Telecommunications Research, one of six national Engineering Research Centers established in 1985 under major grants of the National Science Foundation. In 1986 he was the recipient of the Cooper Union Gano Dunn Award, given annually for outstanding achievement in Science and Technology. In 1989 he received the IEEE Region I Award for outstanding engineering management and leadership.