Small Logs for Transactional Services: Distinction is much more accurate than (Positive) Discrimination *

Debmalya Biswas, Blaise Genest
IRISA/INRIA & CNRS
Campus Universitaire de Beaulieu
35042 Rennes Cedex, France
firstname.lastname@irisa.fr

Thomas Gazagnaire [1]
Citrix Systems R&D Ltd.
Black Horse House, Castle Park
Cambridge, CB3 0FL, United Kingdom
thomas.gazagnaire@citrix.com

Abstract

For complex services, logging is an integral part of many middleware aspects, especially, transactions and monitoring. In the event of a failure, the log allows us to deduce the cause of failure (diagnosis), recover by compensating the logged actions (atomicity), etc. However, for heterogeneous services, logging all the actions is often impracticable due to privacy/security constraints. Also, logging is expensive in terms of both time and space. Thus, we are interested in determining a small number of actions that needs to be logged, to know with certainty the actual sequence of executed actions from any given partial log. We propose two heuristics to determine such a small set of transitions, with services modeled as Finite State Machines. The first one is based on (Positive) Discrimination *of transitions, using every observation to know (discriminate) that a maximal number of transitions occurred. We characterize it algebraically, giving a very fast algorithm. The second algorithm, the* distinguishing *algorithm, uses every observation to maximize the number of transitions which are ensured* not to *have occurred. We show experimentally that the second algorithm gives much more accurate results than the first one, although it is also slower (but still fast enough).*

1. Introduction

An interesting problem for complex systems is to determine a minimal set of actions that needs to be observable such that a given property holds. Some of the properties studied in literature of discrete event systems are normality [8], observability [7], state observability [10], diagnosability [13], etc. Our system corresponds to a (composite) Web service. A Web service [1] refers to an online service accessible via Internet standard protocols. A composite service, composed of already existing (component) services, combines the capabilities of its components to provide a new service. A service schema which specifies the execution order of its components, can be modeled as a Finite State Machine (FSM), performing actions on global variables. We do not tackle here the transformation of a service into a FSM, which should be handled with care to yield a FSM of small size (see [14] and example 1).

Our long-term objective is to provide a transactional framework for (composite) Web services. A transaction [3] can be considered as a group of actions encapsulated by the operations Begin and Commit/Abort, having the following properties: Atomicity (A), Consistency (C), Isolation (I) and Durability (D). Here, we focus on the atomicity aspect, that is, either all the actions of a transaction are executed or none. In the event of a failure, atomicity is preserved by compensation [4]. Compensation consists of executing the compensating actions, corresponding to each executed action of the failed process, in reverse order of the original execution. Thus, for compensation to be feasible, we need to reconstruct each executed action or the complete history of any execution. The usual way of achieving that is to maintain a log of observable actions. However, in addition to the obvious space overhead of logging (in our testing, about 5 times more), the complete log may not always be accessible. For a composite service, the providers of its component services are different. As such, their privacy/security constraints may prevent them from exposing (part of) the logs corresponding to the execution at their sites. Hence, we want from such a partial log to know with certainty the actual sequence of executed actions, to be able to compensate it.

Section 2 introduces the required formal preliminaries including the precise problem statement. Clearly, we are interested in logging the smallest number of transitions possi-

*This work is supported by la Region Bretagne (CREATE ACTIVEDOC) and ANR-06-MDCA-005 DOCFLOW.
[1] Work done while the author was at IRISA, France.

ble. However, finding [15, 9] or approximating [11] the absolute minimal number of such transitions is NP-Complete. We thus propose our first heuristic in Section 3. The idea behind it is that a logged transition allows to *(positively) discriminate* several transitions. That is, we optimize the observable set such that every logged transition implies that a maximum number of transitions are sure to have occurred. We then characterize this algorithm in algebraic terms with a matrix, giving a very fast algorithm (see Section 3.2). Our second idea, presented in Section 4, is that a logged transition also allows to *distinguish* several transitions. That is, we optimize the observable set such that every logged transition implies that a maximum number of transitions *did not* occur. We test both algorithms experimentally in Section 5. The results show that the distinguishing algorithm gives result from 1 (at least as good, which we prove theoretically) to 10 times smaller than the discriminating algorithm, with an average of 1.9 times smaller.

2. Preliminaries

Formally, we model a transactional service as a Finite State Machine (FSM), that is, a 4-tuple $M = (Q, s_0, s_f, \mathcal{T})$, where:

- Q is the finite set of states,
- s_0 and s_f are the initial and final states, respectively,
- $\mathcal{T} \subseteq Q \times Q$ is the (partial) transition relation.

We describe our FSMs as graphs with a unique input and output point, each node and edge corresponds to a state and transition, but we ignore the alphabet. We assume that the service M does not have any unreachable states and that all states can reach the final state s_f. For a state q, $^*q$ (q^*) denotes the set of incoming (outgoing) transitions to (from) q. For convenience, we also assume that there are no outgoing transitions from s_f and no incoming transitions to s_0 (Notice that we could deal with a service without these requirements, but the proof would be more technical.) For a transition $\tau = (q_1, q_j)$, q_i and q_j are referred to as the source and target states of τ. We say that an execution sequence $\rho = \tau_1 \cdots \tau_n \in \mathcal{T}^*$ is a path of M if there exists $q_0, \cdots, q_n \in Q^{n+1}$ with $\tau_i = (q_{i-1}, q_i)$ for all $1 \leq i \leq n$. A path is called initial if furthermore $q_0 = s_0$. We denote by $\mathcal{P}(M)$ the set of initial paths in M. Finally, we denote by $|M|$ the size of M, that is, its number of transitions.

Under restricted observability, for a service $M = (Q, s_0, s_f, \mathcal{T})$, only a subset of its transitions $\mathcal{T}_O \subseteq \mathcal{T}$ are observable, that is, can be logged. In general, for any execution ρ, we call observation projection, the observation we have after ρ was executed (a sequence of transitions, control points, data ..., etc.). We say that an observable projection σ is uncertain if there exists two paths having the same projection. The FSM M is execution sequence detectable iff none of its observable projections are uncertain.

Definition 1 *For an FSM M, let $\mathcal{T}_O \subseteq \mathcal{T}$ be the set of observable transitions. The observation projection $Obs_O : \mathcal{T}^* \longrightarrow \mathcal{T}_O^*$ is the morphism with $Obs_O(a) = a$ if $a \in \mathcal{T}_O$, and $Obs_O(a) = \epsilon$ if $a \in \mathcal{T} \setminus \mathcal{T}_O$, with ϵ the empty word.*

That is, $\mathrm{Obs}_O(\rho)$ is the subsequence of ρ obtained by eliminating from ρ every occurrence of a transition which is not in $\mathcal{T}_O$. With such an observation projection Obs_O, the only way of having execution sequence detectability is to have every transition observable. Indeed, as soon as there exists even one non-observable transition, the service is not execution sequence detectable. Else, let us take a path $\rho\tau$ with the last transition $\tau \notin \mathcal{T}_O$. Then, $\mathrm{Obs}_O(\rho\tau) = \mathrm{Obs}_O(\rho)$. A usual way to overcome such a problem is to ask for certainty only up to the last few events of the sequence [10]. However, this turnaround does not make sense in our framework since if we cannot compensate the very last action, then we cannot compensate any action at all. As such, we design a new observation mechanism, where the last control point reached before failure is monitored, even if the last action is not logged. In practice, it means that every state that is reached is monitored, and overstack the previous state in a special memory buffer.

Definition 2 *Let M be an FSM, $\mathcal{T}_O \subseteq \mathcal{T}$. The observation projection $Obs_O^{last} : \mathcal{T}^* \longrightarrow (\mathcal{T}_O^*, Q)$ is the function $Obs_O^{last}(\rho) = (Obs_O(\rho), q)$ for all $\rho \in \mathcal{P}(M)$ ending in q.*

We will stick with this definition of observability for the rest of the paper. As mentioned before, we are interested in logging as few transitions as possible.

Problem statement. *Given an FSM $M = (Q, s_0, s_f, \mathcal{T})$, we call $\mathcal{T}_O$ an observable set of transitions if the service is execution sequence detectable with Obs_O^{last}. We want to determine an observable set of transitions $\mathcal{T}_O \subseteq \mathcal{T}$ of minimal size. We refer to such a set as a* minimal observable set.

The cardinality of a minimal observable set $\mathcal{T}_O$ of an FSM M is referred to as its observable size $MO(M) = |\mathcal{T}_O|$. Notice that as is usual with decision and computation algorithms, it is sufficient to have an algorithm which from an FSM gives its observable size. That is, we can derive a minimal observable set of the FSM based on an algorithm answering whether its observable size is bigger than n, for any n, and in time polynomially equivalent.

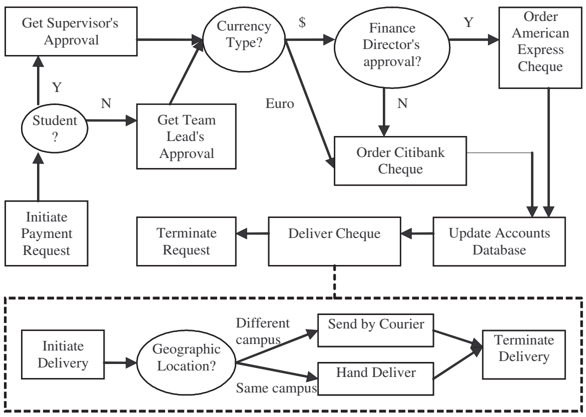

Figure 1. Travel funds request workflow.

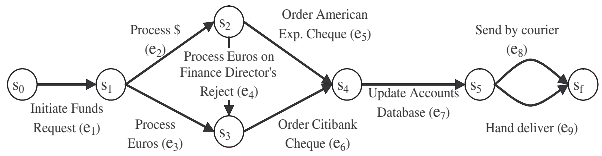

Figure 2. FSM representation of Fig. 1.

Example 1 *We consider in Fig. 1 a travel funds request service, inspired by the workflow in [12]. It involves different departments across organizations.*

We model the service using the FSM $M = (S, s_0, s_f, \mathcal{T})$ representation, as shown in Fig. 2. Notice that this FSM is a simplification of the service, since for instance the choice between the team leader or supervisor approvals is not represented. The reason is that they are both associated with an empty compensating transition, hence knowing which path was taken here is not necessary to be able to perform recovery. However, it is necessary to know which bank issued the cheque in order to able to compensate it, by a "Cancel Last American Express (Citibank) Cheque". It is also possible to handle data being written to the database. For instance, if there is no "Cancel Last Cheque" mechanism, it is possible to force the transition "Update Accounts Database" to be observable, which would lead to the exact amount of the cheque being written to the log, and recovery would manually credit the amount of money written in the log to the corresponding account.

Now, let $\mathcal{T}_O = \{e_2, e_3\}$ and a failure occurs while processing e_8, that is, the cheque is not issued or delivered correctly. Then, $Obs_O^{last}(e_1e_2e_5e_7) = (e_2, s_5) = Obs_O^{last}(e_1e_2e_4e_6e_7)$. Thus, we do not know if an American Express or Citibank cheque was processed. With $\mathcal{T}'_O = \{e_2, e_6\}$, we have $Obs_O^{last}(e_1e_2e_5e_7) = (e_2, s_5)$ and $Obs_O^{last}(e_1e_2e_4e_6e_7) = (e_2e_6, s_5)$, and $\mathcal{T}'_O$ is an observable set of transitions. Notice that every path from s_0 to s_f uses $\mathcal{T}'_O$.

We first relate the problem of computing $MO(M)$ using our definition of observable projections with other known problems. We state now that computing the minimal observable set is equivalent to the *uniconnected subgraph problem*. This problem is also called the *minimal marker placement problem* [9], in the meaning of the following proposition.

Proposition 1 *Let M be an FSM and $\mathcal{T}_O$ a subset of transitions of M. Denote by M' the FSM M obtained by deleting all transitions belonging to $\mathcal{T}_O$. Then, $\mathcal{T}_O$ is an observable set of M iff there does not exist a pair of paths $\rho_1 \neq \rho_2$ of M' with ρ_1 beginning and ending at the same pair of states as ρ_2.*

This problem is NP-complete [9], even with strong restrictions on the graph (acyclic, small indegree and outdegree) [5]. Moreover, this problem cannot be approximated [11] in polynomial time. It means that it is impossible to find algorithms which approximate an observable size close to the absolute minimal size (within a fairly limiting bound), for all FSMs. Anyway, one can try to identify subclasses of FSMs for which this is possible. For instance, hierarchical FSMs are a natural class of FSMs for which the problem can be solved in an efficient way [5]. However, not all FSMs used for modeling services are hierarchical. We propose here fast algorithms which give an observable set of transitions for every FSM. We will then analyze experimentally how far they are from a minimal set, and how often they are far away from the absolute minimal. Indeed, following [11], we know that there will always exist FSMs for which our polynomial time algorithms will be far away from the minimal, but the question which is of interest is how likely it is for such FSMs to occur.

3 (Positively) Discriminating Algorithm

3.1. A Quadratic Time Algorithm

Our first idea is to use the observation as a discriminator. When an observable transition is logged, it allows not only to know that this transition occurred (positive discrimination), but also that other *implied* transitions occurred. That is, the occurrence of some transitions can be deduced from the occurrence of another transition. For example, with reference to Fig. 2, if e_5 occurred, we can also infer that the transitions e_1, e_2, e_7 have occurred. The intuition is to not consider a transition as possibly observable if its occurrence

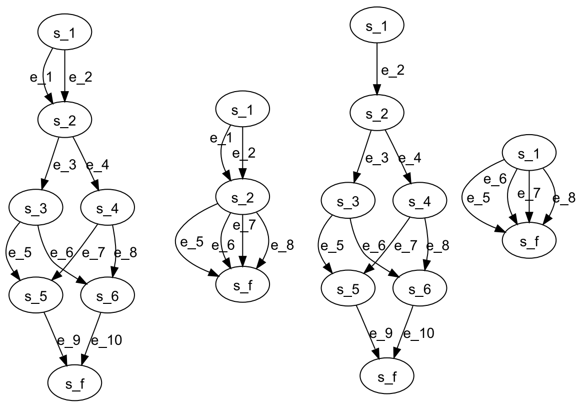

Figure 3. FSM (a), after compression (b), after deleting e_1 (c) and compressing again (d).

can be inferred from the occurrence of other transitions. We thus simply delete such transitions, using the algorithm Compress defined below.

Algorithm Compress FSM $M = (S, s_0, s_f, \mathcal{T})$.
for $i := 1, \ldots, |S|$
  if ($|^*s_i| = 1$ and $|s_i^*| = 1$)
    Let $\tau_1 = (s_j, s_i), \tau_2 = (s_i, s_k)$ for some s_j, s_k.
    Set $\tau_1 = (s_j, s_k)$, delete s_i from S and τ_2 from $\mathcal{T}$
  else if ($|^*s_i| = 1$ and $|s_i^*| > 1$)
    Let $\tau_1 = (s_j, s_i)$ for some s_j.
    For all $\tau = (s_i, s_k) \in s_i^*$, set $\tau = (s_j, s_k)$.
    Delete s_i from S and τ_1 from $\mathcal{T}$.
  else if ($|s_i^*| = 1$ and $|^*s_i| > 1$)
    Let $\tau_2 = (s_i, s_k)$ for some s_k.
    For all $\tau = (s_j, s_i) \in^* s_i$, set $\tau = (s_j, s_k)$.
    Delete s_i from S and τ_2 from $\mathcal{T}$.
  endif
endfor

The FSM M in Fig. 3(a), after compression is shown in Fig. 3(b). It is pretty clear that observing all the remaining transitions $\{e_1, e_2, e_5, e_6, e_7, e_8\}$ is an observable set of M, as shown in the next proposition.

Proposition 2 *Given a service $M = (S, s_0, s_f, \mathcal{T})$, the subset of transitions $\mathcal{T}' \subseteq \mathcal{T}$ obtained on applying algorithm compress to M, is an observable set of transitions of M.*

Proof. We show that for each pair of states $s_1 \neq s_2 \in S$, having more than one distinct path $\rho_1 \cdots \rho_n$ between them, at least one transition of each of the paths is in $\mathcal{T}'$. Without loss of generality, we consider the different cases with respect to ρ_1. Then, we have the following cases:

- ρ_1 is a straight line, that is, all intermediate states in ρ_1 have indegree and outdegree equal to 1. Then, it is easy to see that the outgoing transition of s_1 along ρ_1 would be in $\mathcal{T}'$.

- At least one intermediate state in ρ_1 has indegree > 1. Let us consider the first intermediate state having indegree > 1, say s_i. Traverse backward (towards s_1) from s_i along ρ_1 till we encounter a state having outdegree > 1, say s_j. Then, the subpath of ρ_1 from s_j to s_i is a straight line. And, the outgoing transition of s_j along ρ_1 will be in $\mathcal{T}'$.

- At least one intermediate state in ρ_1 has outdegree > 1. Analogous to the above case.

Notice that the observable set $\mathcal{T}'$ obtained by compression has scope for optimization, as for each pair of states $s_1 \neq s_2$ of M having > 1 paths between them, the transitions in $\mathcal{T}'$ cut *all* paths (as proven earlier). Recall that the existence of one path between a pair of states is not an issue with respect to observability. Thus, instead of selecting all the remaining transitions, let us just select one transition τ_1 and declare it as observable. For instance, in Fig. 3, let $\tau_1 = e_1$. We can delete it from the original FSM since it is positively discriminated (giving Fig. 3(c)). Now, imagine that some state (s_2 in the example) had two incoming transitions before, τ_1, τ_2 ($\tau_2 = e_2$ here). After deletion of τ_1, the state has only one incoming transition, which means that τ_2 will get deleted by the compress algorithm (see Fig. 3(d)). Basically, it means that if we compress, select one transition at a time, delete it from the FSM, and compress again, we end up (in that case) with strictly fewer observable transitions than before (τ_2 is not in the observation set anymore, and we need to observe 4 transitions instead of 5). This gives rise to the following algorithm.

Algorithm Discriminate FSM $M = (S, s_0, s_f, \mathcal{T})$.
Output. An observable set $\mathcal{T}_O$.
Initialization. $\mathcal{T}_O = \emptyset$, $M' = Compress(M)$.
Steps.
while $|M'| > 1$ do
  Select one transition τ of M'.
  Add τ to $\mathcal{T}_0$, delete τ from M.
  $M' = Compress(M)$.
endwhile

Proposition 3 *For a given service $M = (S, s_0, s_f, \mathcal{T})$, the output $\mathcal{T}_O$ of the discriminating algorithm is an observable set of M.*

The problem of this algorithm is that it is in quadratic time: we need to consider every transition twice for each compress step, then we need to call it once for every observed transition. We will now try to characterize the selected observable set of transitions algebraically to obtain an efficient algorithm.

3.2. Algebraic Characterization

Let us define an extension of the classical incidence matrix of a directed acyclic graph (and of a FSM seen as a directed graph), which encodes a directed graph in a matrix. Its first rows express the directed graph, and its last rows express the observation from an observable set of transitions $\mathcal{T}_O$. For convenience, we index the matrix by states and transitions rather than numbers.

Definition 3 (extended incidence matrix) *Let* $M = (Q, s_0, s_f, \mathcal{T})$ *be an FSM and* $\mathcal{T}_O \subseteq \mathcal{T}$ *be a subset of transitions of* M. *The* incidence matrix *of* M, *relative to* $\mathcal{T}_O$ *is a matrix* $A_{M,\mathcal{T}_O}$ *of size* $(|Q| + |\mathcal{T}_O|) \times |\mathcal{T}|$ *defined as follows:*

For every $(s, \tau) \in Q \times \mathcal{T}$*:*

- $A_{M,\mathcal{T}_O}[s, \tau] = 1$ *if* τ *ends in* s*;*
- $A_{M,\mathcal{T}_O}[s, \tau] = -1$ *if* τ *starts from* s*;*
- $A_{M,\mathcal{T}_O}[s, \tau] = 0$*, otherwise.*

For every $(\tau, \tau') \in \mathcal{T}_O \times \mathcal{T}$*:*

- $A_{M,\mathcal{T}_O}[\tau, \tau] = 1$*;*
- $A_{M,\mathcal{T}_O}[\tau, \tau'] = 0$*, if* $\tau' \neq \tau$.

The first $|Q|$ rows of this matrix correspond exactly to the classical incidence matrix. We just append to it a (almost identity) $|\mathcal{T}_O| \times |\mathcal{T}|$ matrix in order to obtain an extended incidence matrix. For example, the matrix corresponding to the FSM M in Fig. 3(a) and $\mathcal{T}_O = \{e_1, e_5, e_6, e_7, e_8\}$, is shown in Fig. 4.

Furthermore, as we translated an FSM $M = (Q, s_0, s_f, \mathcal{T})$ into an algebraic object $A_{M,\mathcal{T}_O}$, we do now transform a path ρ of $\mathcal{P}(M)$ into an algebraic object. Let us denote by $\chi(\rho)$ the vector in $\{0,1\}^{|\mathcal{T}|}$, such that $\chi(\rho)[\tau] = 1$ if τ is fired in ρ, otherwise $\chi(\rho)[\tau] = 0$. Notice that $\chi(\rho)$ characterizes any path ρ of an acyclic graph (the order of transitions can be recovered unambiguously). We now define the observation $V_{M,\mathcal{T}_O}(X)$ associated with a vector X in $\{0,1\}^{|\mathcal{T}|}$ representing a path.

Definition 4 (observation vector) *The* observation *of a vector* X *in* $\{0,1\}^{|\mathcal{T}|}$*, relative to* $\mathcal{T}_O$ *is a vector* $V_{M,\mathcal{T}_O}(X)$ *of size* $|Q| + |\mathcal{T}_O|$*, such that for every* $s \in Q$

- $V_{M,\mathcal{T}_O}(X)[s] = -1$ *if* X *starts from* s*;*
- $V_{M,\mathcal{T}_O}(X)[s] = 1$ *if* X *ends in* s*;*

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	e_{10}
s_1	-1	-1	0	0	0	0	0	0	0	0
s_2	1	1	-1	-1	0	0	0	0	0	0
s_3	0	0	1	0	-1	-1	0	0	0	0
s_4	0	0	0	1	0	0	-1	-1	0	0
s_5	0	0	0	0	1	0	1	0	-1	0
s_6	0	0	0	0	0	1	0	1	0	-1
s_7	0	0	0	0	0	0	0	0	1	1
e_1	1	0	0	0	0	0	0	0	0	0
e_5	0	0	0	0	1	0	0	0	0	0
e_6	0	0	0	0	0	1	0	0	0	0
e_7	0	0	0	0	0	0	1	0	0	0
e_8	0	0	0	0	0	0	0	1	0	0

Figure 4. Matrix of the FSM M in Fig. 3(a) and $\mathcal{T}_O = \{e_1, e_5, e_6, e_7, e_8\}$.

- *otherwise* $V_{M,\mathcal{T}_O}(X)[i] = 0$.

Moreover, for every $\tau \in \mathcal{T}_O$*, we have:*

- $V_{M,\mathcal{T}_O}(X)[\tau] = 1$ *if* $X[\tau] = 1$*;*
- *otherwise* $V_{M,\mathcal{T}_O}(X)[\tau] = 0$.

Clearly, $\mathcal{T}_O$ is an observable set of transitions iff there *does not* exist $X \neq Y$ with $V_{M,\mathcal{T}_O}(X) = V_{M,\mathcal{T}_O}(Y)$. We use now the algebraic characterization.

Proposition 4 *Let* $X, Y \in \{0,1\}^{|\mathcal{T}|}$. *Then we have:*

$$V_{M,\mathcal{T}_O}(X) = V_{M,\mathcal{T}_O}(Y)$$

$$\Leftrightarrow$$

$$A_{M,\mathcal{T}_O} \cdot Y = V_{M,\mathcal{T}_O}(X) = A_{M,\mathcal{T}_O} \cdot X$$

Using proposition 4, we obtain that $A_{M,\mathcal{T}_O}$ *is injective implies that* $\mathcal{T}_O$ *is an observable set of transitions*. It can be shown that the discriminating algorithm gives a minimal set for which $A_{M,\mathcal{T}_O}$ *is injective*, thus giving us the algebraic characterization we were looking for (we also checked that fact experimentally).

Now, it is well known that a matrix is injective iff its kernel has dimension 0. In our case, it is also well known [6] that the kernel of the incidence matrix $A_{M,\emptyset}$ corresponds exactly to the so-called cycle space of M. Moreover, the dimension of the cycle space (also known as the cyclomatic number) of a graph with n vertices, m edges and K connected components (considering edges are unoriented) is exactly $m-n+K$. That is, it suffices to observe $m-n+K$ edges to have an injective matrix (and equivalently an observable set of transitions). That is, $MO(M) \leq m-n+K$.

3.3. Analysis of the Matrix Algorithm

The matrix algorithm is clearly very fast to give an approximation of $MO(M)$, since the starting graph is connected. It suffices to count $m-n+1$ to know the number of observable transitions output by the discriminating algorithm, which is an immediate number to compute. For instance, in the example of Fig. 3, $10-7+1=4$. Knowing the transitions to log is not much harder to compute either. It suffices to run the algorithm incrementally, selecting one transition at a time. Let us now analyze which transitions are important to log. Assume we are choosing to log some transition τ. Either deleting τ increases the number of connected components of M or not. If it increases the number of connected components, logging it leads to the matrix algorithm on the resulting FSM giving us the transition to observe, plus $m'-n'+K'$, where m', n', K' are the values for the new FSM, that is, $1+m'-n'+K' = 1+(m-1)-n+(K+1) = m-n+K+1$. Clearly, logging this transition is not a good idea since it increases the number of transitions to log in the end. Hence, we shall never log a transition whose deletion increases the number of connected components.

On the other hand, if deleting τ keeps the number of connected components constant, then $1+m'-n'+K' = 1+(m-1)-n+K = m-n+K$, and the transition we are choosing is useful. That is, any set of $m-n+1$ transitions whose deletion keeps the FSM connected is an observable set of transitions. To sum up, we know that we can select any transition, as long as its deletion does not disconnect connected components. Choosing with care the transition we select may optimize the result. For that, we need some heuristical metrics to tell us which transition is or is not good to observe.

Let us analyze the algorithm. Notice first that thinking in terms of positive discrimination is not always good. Indeed, in the example of Fig. 3, the compressed versions are shown in Fig. 3(b and d), leading to the observable set $\{e_1, e_5, e_6, e_7\}$. However, a minimal observation set is $\{e_1, e_4, e_{10}\}$, and observing any transition in e_5, e_6, e_7, e_8 leads to observing at least 4 transitions in order to get an observable set.

We call initial (final) state a state which has no incoming (outgoing) transitions. In the matrix algorithm, there can be more than one initial and/or final states. Indeed, in Fig. 3, if we delete e_4, then s_4 becomes an initial state. However, the matrix algorithm does not consider which transition is reachable from which initial state, and which final state it can reach. In particular, it is possible that $m-n+K \neq 0$, but that the FSM is observable with the current set of observable transitions (no more transitions are needed), while the matrix algorithm still asks to select some more. For example, with reference to Fig. 3, after deletion of $e_1, e_4, e_{10}, m-n+1 \neq 0$, but $\{e_1, e_4, e_{10}\}$ is already an observable set.

4 Distinguishing Algorithm

Based on the second observation, we design the following method to select transitions to be observed:

1. Select a transition e whose deletion does not disconnect the FSM.

2. Ensure that there is an initial state s_0, a state s and two paths between (s_0, s), one using e and one not using e (if it is not the case, then this transition is useless for observability).

With such a method, we are sure to get at most as many transitions as observed by the matrix algorithm. Now, notice that the first condition is actually implied by the second condition. Indeed, if there exists two states s, t, two paths $e_1 \cdots e_m$, $f_1, \cdots f_n$ and $e_i \notin \{f_j \mid j \leq n\}$, then it means that deleting e_i cannot disconnect the FSM, since the undirected path $e_{i-1} \cdots (e_1 = f_1) \cdots (f_n = e_m) \cdots e_{i+1}$ allows to indirectly connect $e_{i-1}{}^* =^* e_i$ and ${}^*e_{i+1} = e_i^*$. So, we can just delete the first condition. Here is the algorithm we use for testing the second condition in an efficient manner, based on Depth First Search (DFS). Basically, the algorithm breaks and returns t as soon as one transition t points to a state s' which has been previously explored (that is, there is another path, not using t which reaches s' from an initial state s from which t is reachable. That is, we have two paths connecting s to s', one using t and one not using t). DFS keeps a stack S, a hash table H of states which have already been explored by the search, and each transition is tagged as explored or unexplored. $S.head$ designates the head of the stack, and $t.dest$ the target state of a transition t.

Algorithm Test.
Create Hash table H.
for each initial state s_0 of M.
  Initialize H to empty, S to s_0.
  Set all the tags to unexplored.
  Run DFS from s_0:
  while S is nonempty do
    while there is an unexplored transition t
        from state $S.head$ do
      Tag t as explored.
      if $t.dest \in H$, *then* return t.
      else insert $t.dest$ into H and push $t.dest$ on S.
      endif
    endwhile
    pop S
  endwhile

endfor
return "No more transitions to explore".

Now, notice that there can be plenty of transitions which satisfy condition 2. We would like to have a more precise metric to choose the transition to observe in that set. Rather than optimizing the number of transitions we are positively discriminating, we want now to optimize the number of transitions we distinguish the transition from. That is, we want to optimize the number of other paths not using that transition, but connecting two states which are also connected with a path using that transition. However, computing exactly that number of paths would be really inefficient. We thus propose an efficient but slightly less accurate version: we want to maximize the number of initial or final states from or to which the second condition is true. It suffices simply to modify the previous algorithm, by keeping a counter associated with each transition, and not breaking when a transition hitting H is seen, but by increasing the counter of that transition. Furthermore, we also increment the counter of the first transition which led to the insertion of that state into H initially. The counter of a transition is increased at most once per initial state. We call this new algorithm *Count*. Obviously, an algorithm *CountBack* can be similarly devised, running from all final states and traversing transitions backwards. Now, our new distinguishing algorithm proceeds as follows:

Algorithm Distinguish FSM $M = (Q, s_0, s_f, \mathcal{T})$.
Create Set $\mathcal{T}_O$.
loop
  Set all transition counters to 0.
  Run at random Count or CountBack.
  if all transition counters are 0, *then* return $\mathcal{T}_O$.
  else select one transition t with maximal counter value.
    Add t to $\mathcal{T}_0$.
    Delete t from $\mathcal{T}$.
  endif
endloop

The previous claims we made allow easily to state the following.

Proposition 5 *For an FSM $M = (Q, s_0, s_f, \mathcal{T})$, the distinguishing algorithm returns an observable set $\mathcal{T}_O$ of transitions. Moreover, its size is always $|\mathcal{T}_O| \leq (|\mathcal{T}| - |Q| + 1)$, which is the observable size returned by the matrix algorithm and the positively discriminating algorithm.*

5. Experimental Evaluation

We have some theoretical clues about how our algorithms fair against each other, and how close they can approximate the absolute minimal observable size. Because our matrix and distinguishing algorithms are polynomial time, we know that there are FSMs on which they give an answer far away from the optimum [11]. The question is how far they are, and how often it happens. The second fact is that the distinguishing algorithm gives a set never bigger than the one given by the matrix algorithm. The question then is: is it better, and if yes, by how much and how often is it much better.

No. of	Observable size		
edges	Absolute	Matrix	Disting.
42	9	9	9
58	14	14	14
75	18	19	18
103	20	24	20
102	32	33	32
123	31	33	32
146	33	34	34
146	42	42	42
178	36	37	36
185	38	38	38
223	43	47	44
221	53	57	55
241	57	62	58
273	51	57	53
280	72	75	72
294	74	76	75
325	69	78	73
326	75	79	76
355	74	81	77
345	88	91	88
382	84	90	85
387	92	96	94
410	81	87	83
445	89	97	90
448	100	106	104
460	101	108	105
484	99	107	103
489	104	111	107
540	106	122	112
550	93	101	95
570	108	118	111
605	108	124	111
618	121	136	125
634	120	131	124
631	115	121	118
657	131	141	134
672	134	146	139
699	126	141	131
704	142	156	150

No. of	Obs. size	
Edges	Matrix	Distinguishing
97	41	11
127	55	13
85	29	19
115	47	20
137	52	23
126	43	27
264	120	20
312	143	22
173	63	36
201	77	52
431	197	28
452	205	40
103	22	17
200	72	39
133	36	25
356	147	74
301	131	45
114	27	19
785	367	40
169	45	30
98	16	16
101	17	17
175	49	34
987	462	72
132	29	24
490	203	113
464	188	110
158	37	27
115	17	17
620	268	140
121	17	17
631	268	141
1013	468	90
128	18	18
165	53	37
114	56	56
214	58	36
137	19	19
508	223	86
228	53	39
747	315	178
222	55	38
947	416	217
790	340	219
161	23	23
1023	447	257
225	46	34
997	431	230
311	89	60
253	58	40
923	419	106
321	90	63
182	24	24
1110	480	276

Figure 5. Raw data for hierarchical (left) and general (right) FSMs.

The first question is difficult to answer accurately, since obtaining the absolute minimal observable size is intractable. One solution could be to look at small enough FSMs to get the values, but the problem is a variation of one observable transition having a big impact percentage wise in small sets, so the results would not be very meaningful. Instead, we focus on particular non-trivial FSMs, namely hierarchical FSMs [2]. For instance, the system in Fig. 1 is hierarchical, with 2 components. In [5], we present a polynomial time divide and conquer algorithm to compute the minimal observable size of a hierarchical FSM based on the observable sizes of its components. This allows us to compute the absolute minimal observable size of large hierarchical FSMs, as long as the components are small enough. We thus performed our two heuristics plus the absolute minimal algorithm on hierarchical FSMs, giving the results on the left part of Fig. 5, analyzing the data in the next section. Furthermore, to confirm or infirm the conclusions we draw on hierarchical FSMs, we also performed experiments for our two heuristics on general FSMs, whose results are given on the right part of Fig. 5.

5.1. Hierarchical FSMs

We generate hierarchical FSMs randomly, using the following method for each FSM. First, we choose a number (between one and forty) of base subcomponents in the FSM. Then, we generate each of them randomly by using the *Synthetic DAG Generation Tool* (http://www.loria.fr/˜suter/dags.html), varying randomly the input parameters, to get FSMs as diverse as possible. We then generate inductively a hierarchical FSM having these base components. On those FSMs, we run the algorithm from [5] to get the absolute minimal observable size, as well as the matrix and distinguishing algorithms. Fig. 6 shows the result we obtained, according to the number of transitions of the global FSM.

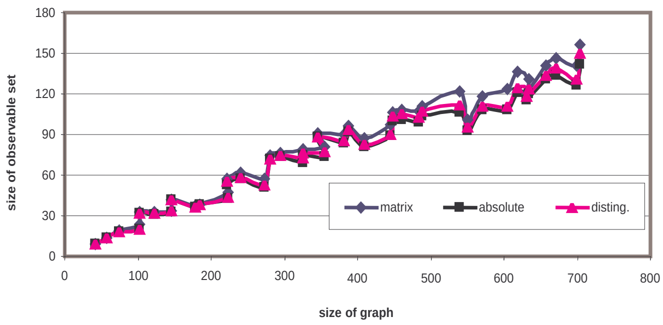

Figure 6. Observable size vs. number of transitions, over 40 randomly generated hierarchical FSMs.

The graph confirms that the matrix algorithm gives worse results than the distinguishing algorithm, which gives worse results than the absolute algorithm, but the difference does not seem very important.

Let us analyze more precisely the percentage difference between the absolute minimal observable size and the ob-

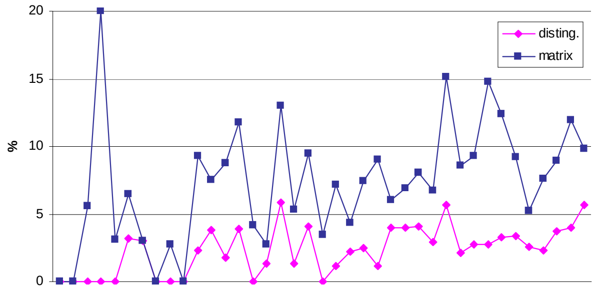

Figure 7. Deviation in percentage from the absolute minimal observable size over 40 randomly generated hierarchical FSMs.

servable sizes given by the matrix and distinguishing algorithms, on the same data (see Fig. 7). It seems that the distinguishing algorithm comes much closer to the absolute minimum, from 0% to 6%, 2% in mean value. The matrix algorithm is sometimes as good as the absolute minimum, sometimes much worse (20% more transitions need to be logged), 6% in mean value, that is 3 times more than the distinguishing algorithm.

Last, we can analyze the percentage of transitions logged by the different algorithms (see Fig. 8). As mentioned in the beginning, this number is quite close for the 3 algorithms, ranging from 17% to 33%. In mean values, the algorithms needs to log 20%, 21% and 22% of transitions, respectively. In terms of time taken, the matrix algorithm is instantaneous for our biggest FSM (700 transitions), the distinguishing algorithm takes 2.5 seconds to finish, and the absolute minimum takes half an hour.

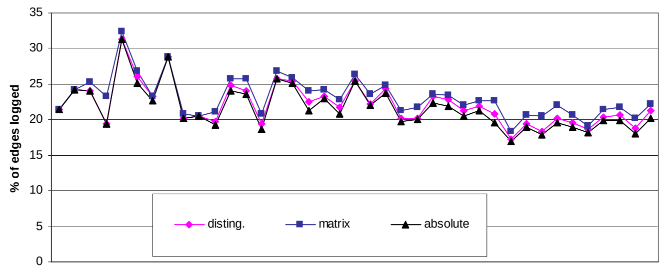

Figure 8. Percentage of edges logged by the different algorithms, over 40 randomly generated hierarchical FSMs.

5.2. General FSMs

As mentioned previously, our first analysis is made only on particular FSMs, hence no general conclusions can be made by considering only hierarchical FSMs. We now turn to more general FSMs (on which however we cannot know the absolute minimum), to get a clue whether our first conclusions are true or not. We again use the *Synthetic DAG Generation Tool*, with random parameters for each size of FSM from 80 to 1200 transitions.

Fig. 9 shows the result we obtained using the matrix and distinguishing algorithms, according to the number of transitions of the FSM. The graph shows a much more chaotic picture than the one obtained on hierarchical FSMs. Furthermore, the distinguishing algorithm seems to often do much better than the matrix algorithm. Still, there are several cases (around 100 transitions) where both give the same results. Concerning time, the distinguishing algorithm takes at most 15 seconds to perform (remark that the time taken is proportional to the number of transitions logged rather than to the number of transitions in the FSM).

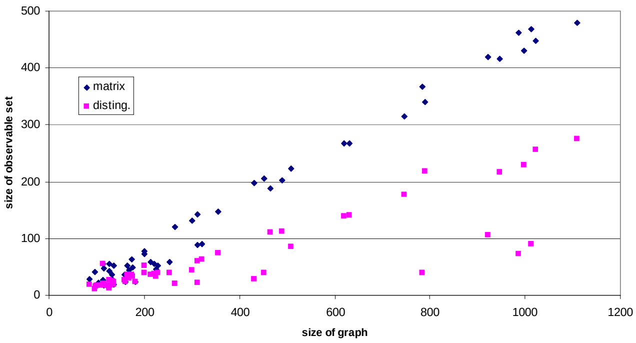

Figure 9. Observable size vs. number of transitions, over 60 randomly generated general FSMs.

Let us now analyze the percentage of transitions logged by the different algorithms (Fig. 10). Again, we see the chaosness of the picture, ranging from 15% to 50% of transitions logged by the matrix algorithm (mean value 34%), and from 4% to 50% for the distinguishing algorithm (mean value 18%). The comparison with results obtained on hierarchical FSMs (Fig. 8) is quite interesting. The percentage of transitions can vary from 1 to 10, while it was from 1 to 2 in the hierarchical case. The matrix does much worse in mean value (34% vs 22%), which is understandable since the FSMs are less regular and more complicated than in the hierarchical case. On the other hand, the distinguishing algorithm succeeds slightly better on this unrestricted FSMs than on hierarchical FSMs (18% vs 21%).

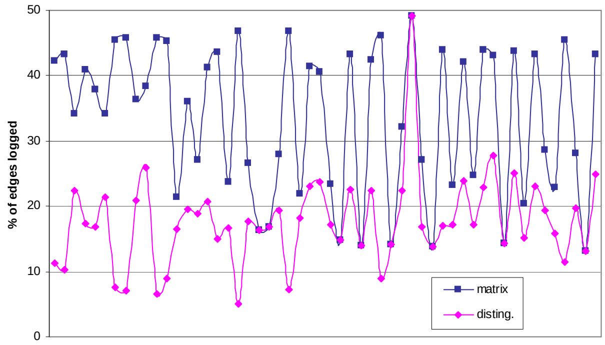

Figure 10. Percentage of edges logged by the matrix and distinguishing algorithms, over 60 randomly generated general FSMs.

Finally, we give a summing up graph in Fig. 11, where we put each random FSM we generated according to the percentage of transitions logged by the matrix algorithm and by the distinguishing algorithm, together with two broken lines labelled by 18% (vertically for the distinguishing algorithm) and 34% (horizontally for the matrix algorithm) showing the mean values of the percentage of transitions logged. On this graph, we can draw the line (broken line labelled 100%) on which both algorithms perform similarly. We can see that it happens several times, but mainly when the matrix algorithm already gives good results (from 12% to 18% of transitions logged, which shall be close if not equal to the optimal). Only once, the matrix is bad and so is the distinguishing algorithm (around 50% of transitions logged). It was to be expected that such cases occur, since we know that the absolute minimal is not approximable, but luckily, it is pretty rare.

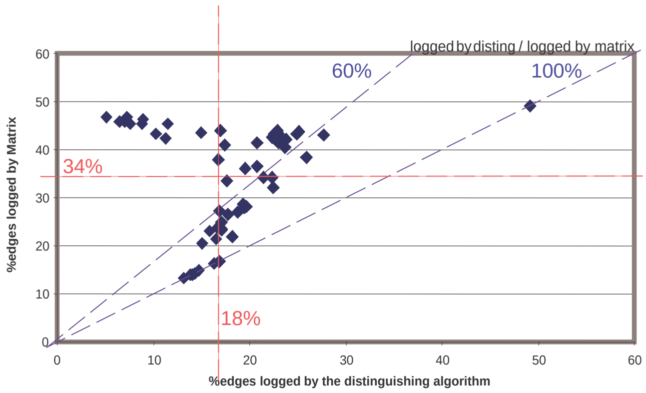

Figure 11. Percentage of edges logged by the matrix vs. distinguishing algorithm, over 60 randomly generated general FSMs.

Overall, the distinguishing algorithm gives an observable size 0.6 times the size returned by the matrix one (we draw a broken line labelled by 60% to separate the experiments under and over that value), and almost all of its answers are within 0.7 times of the matrix algorithm. Moreover, it sometimes gives one tenth the number of transitions to log compared to the matrix algorithm (which implies that the matrix algorithm can be very inaccurate). Also, notice that only once, the distinguishing algorithm gives more than 30% of transitions to log (1.5% of the FSMs), while it is the case for 50% of the FSMs in the matrix algorithm.

6. Conclusion

We proposed two polynomial time algorithms to get an (over) approximation of the minimal number of actions to log in composite services to be able to compensate it. We modeled the services as FSMs. Our first algorithm, based on an algebraic characterization, is very fast, though it can be imprecise (in several cases, it gives at least 10 times as many transitions to log compared to the absolute minimal size). Our second algorithm based on a heuristic trying to distinguish as many conflicting transitions as possible with an observable transition, is slower but still efficient (we don't need more than 15 seconds for 1200 transitions), and usually gives much smaller observable sets. Still, in one case, it seems to give inaccurate results. There are probably some more heuristics to apply to get a more accurate algorithm. Nevertheless, in mean value, it seems that the distinguishing algorithm gives results close to the absolute minimum (18% of transitions, while we get 20% for the absolute minimal, looking at hierarchical FSMs), so efforts to optimize it further would probably not be worth it but for very few pathological cases, and would slow down the algorithm.

References

[1] G. Alonso, F. Casati, and H. Kuno. *Web Services: Concepts, Architecture and Applications*. Springer, ISBN: 3540440089, 2004.

[2] R. Alur and M. Yannakakis. Model checking of hierarchical state machines. *ACM Transactions on Programming Languages and Systems (TOPLAS), 23(3)*, pages 1–31, 2001.

[3] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, ISBN: 0201107155, 1987.

[4] D. Biswas. Compensation in the world of web services composition. *In proceedings of the International Workshop on Semantic Web Services and Web Process Composition (SWSWPC), LNCS 3387*, pages 69–80, 2004.

[5] D. Biswas and B. Genest. Minimal observability for transactional hierarchical services. *In proceedings of the 20th International Conference on Software Engineering and Knowledge Engineering (SEKE)*, pages 531–536, 2008.

[6] C. Godsil and G. Royle. *Algebraic Graph Theory*. Springer, ISBN: 0387952209, 2001.

[7] R. Kumar and V. K. Garg. *Modeling and Control of Logical Discrete Event Systems*. Springer, ISBN: 0792395387, 1994.

[8] F. Lin and W. M. Wonham. On observability of discrete-event systems. *Information Sciences, 44(3)*, pages 173–198, 1988.

[9] S. Maheshwari. Traversal marker placement problems are np-complete. *Boulder University Research Report CU-CS-092-76*, 1976.

[10] C. M. Özveren and A. S. Wilsky. Observability of discrete event dynamical systems. *IEEE Transactions on Automatic Control, 35(7)*, pages 797–806, 1990.

[11] K. Rohloff, S. Khuller, and G. Kortsarz. Approximating the minimal sensor selection for supervisory control. *Discrete Event Dynamic Systems, 16(1)*, pages 143–170, 2006.

[12] W. Sadiq and M. E. Orlowska. Analyzing process models using graph reduction techniques. *Information Systems, 25(2)*, pages 117–134, 2000.

[13] M. Sampath, R. Sengupta, S. Lafortune, K. Sinaamohideen, and D. Teneketzis. Diagnosability of discrete event systems. *IEEE Transactions on Automatic Control, 40(9)*, pages 1555–1575, 1995.

[14] A. Wombacher, P. Fankhauser, and E. Neuhold. Transforming bpel into annotated deterministic finite state automata for service discovery. *In proceedings of the IEEE International Conference on Web Services (ICWS)*, pages 316–323, 2004.

[15] T.-S. Yoo and S. Lafortune. Np-completeness of sensor selection problems arising in partially-observed discrete-event systems. *IEEE Transactions on Automatic Control, 47(9)*, pages 1495–1499, 2002.