# Automating the Discovery of AS-IS Business Process Models: Probabilistic and Algorithmic Approaches

Anindya Datta
*DuPree College of Management, Georgia Institute of Technology, 755 Ferst Drive, Atlanta, Georgia 30332-0520*
*adatta@cc.gatech.edu*

In the current corporate environment, business organizations have to reengineer their processes to ensure that process performance efficiencies are increased. This goal has lead to a recent surge of work on *Business Process Reengineering* (BPR) and *Workflow Management*. While a number of excellent papers have appeared on these topics, all of this work assumes that existing (AS-IS) processes are known. However, as is also widely acknowledged, coming up with AS-IS process models is a nontrivial task, that is currently practiced in a very ad-hoc fashion. With this motivation, in this paper, we postulate a number of algorithms to discover, i.e., come up with models of, AS-IS business processes. Such methods have been implemented as tools which can automatically extract AS-IS process models. To the best of our knowledge, *no* such work exists in the BPR and workflow domain. We back up our theoretical work with a case study that illustrates the applicability of these methods to large real-world problems. We draw on previous work on process modeling and grammar discovery. This work is a requisite first step in any reengineering endeavor. Our methods, if adopted, have the potential to severely reduce organizational costs of process redesign.
(*Workflow Management; Business Process Reengineering; AS-IS Business Process Models; Process Discovery; Algorithms*)

## 1. Introduction, Research Context, and Related Work

The related areas of *Workflow Management* and *Business Process Reengineering* (BPR) have attracted much research attention in the recent past. A lot of this work is concerned with *business process redesign and innovation* (Hammer and Champy 1993, Davenport 1993, Hammer and Stanton 1995, Yu 1995, Dinkhoff et al. 1994a). Process redesign is perceived to bring about major improvements in the achievement of organizational objectives such as service level and quality, by reducing process cost and time. For anyone unfamiliar but interested in this area (i.e., BPR, workflows, business processes etc.) we would recommend two citations—Davenport (1993) and Hammer and Champy (1993). These works may be regarded as the "pioneering" literature in this area.

To perform BPR, one must first come up with models of existing processes. More specifically, it is well-recognized that models and documentation of existing processes are prerequisites for BPR participants in understanding the existing state of the organization, and learning about existing problems that have to be avoided in the "innovated" environment (Davenport 1993, Dinkhoff et al. 1994a). Such models are known as AS-IS process models. As definitively stressed in both Davenport (1993) and Hammer and Champy (1993), the modeling of AS-IS processes is the requisite

1047-7047/98/0903/0275$05.00

starting point of any reengineering endeavor. Current BPR research implicitly assumes that the AS-IS model of organizational processes are always known prior to reengineering. However, current work also recognizes that AS-IS process models are very difficult (not to mention extremely expensive) to extract (Davenport 1993, Hammer and Champy 1993). Furthermore, this same literature characterize existing procedures for modeling AS-IS processes as ad-hoc, usually consisting of extensive meetings and discussions with managers and employees individually and in groups. Typically, business processes consist of *tasks* performed by agents. Individual agents often know what they do, but have very little idea of what happens after their task is performed. On the other hand, managers usually have a high level understanding of the process, e.g., the interagent flow, but are often unaware of individual tasks.

As an example, consider the process of travel request approvals at universities. After the travel request is initiated by a faculty member by filling out a form, it is processed by a departmental secretary before being routed to the university travel support office for further processing and approval. Often the faculty member does not know what the secretary does to process her request, and the secretary has very little idea how the travel support office deals with his output. Similarly, the travel support office supervisor, while knowing the basic flow of information, does not know what each individual agent does. Thus, extracting the AS-IS process model of travel request processing would require extensive meetings with each individual agent, as well as group meetings with all performers and managers. Usually, large consulting organizations spend weeks and charge many thousands of dollars for these types of tasks (Davenport 1993). Also, it is easily seen that this task is an unavoidable one, as, until existing processes are modeled, they cannot be reengineered. Moreover, the process model derived in such a fashion has no rigor or formalism behind it—two different consulting organizations are apt to come up with very different models for the same process. Therefore, as there exists no standard procedure to model AS-IS processes, most organizations are at the mercy of specialized individuals (e.g., consultants) who come at a heavy cost.

There is, unquestionably, a clear-cut need to come up with formal, rigorous ways of discovering AS-IS models for business processes. The major benefit of such methods is that they can then be implemented as tools, which could be used by "lay persons" to get a reasonable first cut of existing processes. While the utility of business process modeling and redesign is widely accepted, we are aware of *no work* that describes strategies for *systematic and automated* extraction of AS-IS process models. Our primary objective in this research is to propose a framework and methodology for automated AS-IS business process discovery. In other words, we propose to design strategies that will enable the programmatic discovery of business processes. The basic approach we adopt is to observe the *behavior* of a process, and extract a formal model of the process that can account for this behavior. There exist "tried and true" methods to observe and record process behavior. We use these methods to record *traces* of process behavior. Then we show that the problem of process discovery from these traces maps to the problem of *grammar discovery* from examples of sentences in a regular language. The grammar discovery problem has a solid theoretical basis, which we subsequently exploit to come up with rigorous process discovery algorithms. A case study reveals that our process discovery methods work quite well in practice. Also, since our algorithms are automated, they take far less time than the currently used ad-hoc procedures. If employed in practice, our approach has the potential to save organizations large sums of money from their BPR budget. If current trade literature is to be believed (e.g., *Datamation, Computer World*) such costs for mid-size organizations (100 to a 1,000 employees) run into millions of dollars a year.

In this paper we suggest three different strategies that may be used to extract AS-IS process models. These strategies are based on well-studied methodologies, namely *stochastic modeling* and *finite state machine synthesis*. Because our strategies are systematic (i.e., procedural) they can be coded into programs. We have done so and have used these programs to extract models from existing "real-life" processes. Such extraction

is done using substantially less time and resources than conventional ways.

Work in the following areas are used as basis of our approach:

1. Work in *process modeling*. This work lays the foundation of the process model we use in this paper described in §2. While looking at process modeling literature, we looked at two substreams, (a) *Business Process Modeling and Workflows* (Blyth et al. 1993, Medina-Moza et al. 1992, Yu 1995, Yu and Mylopoulos 1994, Yu et al. 1995, Rajapakse and Orlowska 1995, Anton et al. 1994), and (b) Software Process Modeling (Dinkhoff et al. 1994b, Kamath and Ramamritham 1996, Kuo et al. 1996, Ngu et al. 1996, Rusinkiewicz and Sheth 1995, Weissenfels et al. 1996).

2. Work in *grammar discovery* (see Angluin and Smith (1983) for a comprehensive survey). In Cook and Wolf (1995) there is a nice application of these methods to software process discovery. We borrow from this work to devise our strategies. However, as explained later, substantial modifications need to be made to apply these methods to our problem.

The rest of the paper is organized as follows. Section 2 postulates a model of business processes, followed by an overview of our basic ideas to discover process models in Section 4. In Sections 5 and 6 we state, with examples, three different algorithms that we have designed. We discuss the effectiveness of these algorithms in Section 7. Subsequently, we provide a detailed case study in Section 8 to illustrate the applicability of our methods to a large scale problem and conclude in Section 10.

## 2. Organizational Process Model

Before describing strategies to extract models of business processes (BP), we believe it is necessary to postulate what we mean by a business process and how we propose to model it. In particular we wish to propose a simple, yet comprehensive model of a BP, which would then be used in the rest of the paper. First we give a brief preview of this section: to describe a BP, we use the notions of *events*, *states*, and *activities*. We first define these terms. Note that the idea of describing processes in terms of the above notions has been widely used in the process modeling literature (Sato and Praehofer 1997, Cook and Wolf 1995). Having defined and explained these notions we represent a BP in terms of a structure that we call a *Process Activity Graph* (PAG). Thus, the problem of discovering an AS-IS process model maps to coming up with a PAG for a BP.

To illustrate the discussion in this section, we use the following organizational scenario as a running example: The Information Technology Services Unit (ITSU) of a department at a university. The ITSU offers computing facilities housed in a Computing Lab. The ITSU also provides maintenance support and consultancy to the faculty, staff, and students of the department. It also maintains a computer-equipment repository, and periodically loans equipment such as computers and LCD projectors to the users.

The ITSU computing Lab serves as the coordination center of all these tasks, and as the official Help Desk for all user support requests. The key active participants of the ITSU are:

- The Lab Technicians, whose duties include computer maintenance service and consulting.
- The Lab Monitors, who supervise the Lab, act as liaisons between users and the ITSU, and maintain the equipment-loan system.
- The Lab Manager, who supervises all the technicians and monitors, organizes the important ITSU activities such as computing workshops, and publishes the ITSU newsletter.

All the users of the ITSU services communicate with the Lab Monitor, who then routes the messages and requests to the appropriate ITSU employees. In some cases, the Monitor may be able to address the request without any assistance from others. He/she is also responsible for distributing ITSU information brochures and conveying important announcements to the users. Using this scenario as an example, we now define the notions of *agents*, *events*, *states*, and *activities*. These definitions are adapted from similar definitions found widely in the process modeling literature.

DEFINITION. **Agent.** An agent is an active organizational participant.

Agents may be looked upon as performers of organizational tasks. In the ITSU example, the agents include the Lab Monitor, the Manager, and the Lab Technicians.

DEFINITION. **Event.** An event is an observable, instantaneous occurrence of organizational significance and indicates the beginning or end of organizational tasks.

The phrase "organizational significance" simply implies that an event, from a process modeling perspective, must somehow be related to the organization under observation. For example, for an analyst attempting to model loan processing in a bank, the occurrence *atmospheric temperature exceeds hundred degrees* is not relevant, and therefore is not regarded as an *event*. Furthermore, in modeling business processes, we are not interested in any organizational occurrence, but only in those that mark the start or end of tasks. We clarify this point further after defining the notion of an *activity* below. Each event is described as a 2-tuple $[L, TS]$, where $L$ is a unique event label and $TS$ is a timestamp of the event. Since events denote the initial and terminal points of tasks, we now proceed to model tasks in terms of *activities*.

DEFINITION. **Activity.** An activity is a logical unit of work performed by a single agent, that occurs over a temporally extended period. An activity is initiated by a begin_event, which signals the start of the activity and terminated by an end_event, which signals the end of the activity.

Any activity $A$ is characterized by a 4-tuple $[L_A, G, E_{beg}, E_{end}]$, where $L_A$ is an activity label, $G$ is the agent who performed the activity, and $E_{beg}$ and $E_{end}$ denote the begin and end events for the activity. The temporal duration of an activity is delimited by the timestamps of its begin and end events i.e., duration $(A_i) = TS(E_{end}) - TS(E_{beg})$. Since, in our model, we use events to characterize the begins and ends of activities, we use the following simple naming convention for event labels: an event that serves as the begin event for activity $A$ is denoted by $A^+$ and the end event for activity $A$ is denoted by $A^-$.

In our ITSU scenario, an example activity may be *Receive Support Request* (*R-SR*). This activity would denote the organizational task performed in receiving a request for technical support from some user. As described earlier, this activity would be characterized by a begin event $R\text{-}SR^+$ and an end event $R\text{-}SR^-$.

Having described activities, we now turn our attention to the effect of performing an activity upon an organization. Essentially, activities result in the alteration of the organizational *state*. The process of activities changing organizational states may be viewed as analogous to transactions changing database states.

DEFINITION. **State.** The state of an organization at a given instant in time is the situation of the organization at that time. One may also conceptualize state as a snapshot of an organization at a given time.

Having defined activities and states, we are now in a position to present a model of a business process (BP). While many definitions of BPs exist in the literature, we adapt the following definition from the well-known work by Davenport (1993, p. 5).

DEFINITION. **Business Process.** A business process is a specific ordering of work activities, across time and place, with a beginning, an end and clearly identified inputs and outputs.

A BP is initiated with respect to an input state of the organization, and subsequently, through a sequence of activities, terminates at one or more outputs or *goal* states of the organization. In between the terminal states, a BP may take the organization through a succession of intermediate states. We hasten to add that this way of characterizing processes, i.e., using the notions of activities and states, is widely used in the process modeling literature. For example, software processes have been modeled in the same way in Cook and Wolf (1995), while similar models of business processes may be found in Sato and Praehofer (1997). Our goal is not to stipulate a novel process modeling methodology. Rather, we wish to state a simple yet comprehensive model of a BP using established notions. Subsequently, using this established model, we will design novel ways to discover processes.

Having described the model of a BP, we turn our attention to stating a simple representation of such a model. Again, established literature comes to our aid. Process models have often used graph structures to represent activity sequences. Accordingly we represent a BP using a *Process Activity Graph* (PAG) as defined below.

DEFINITION. **Process Activity Graph (PAG).** A PAG is a 2-tuple $[S,A]$, where $S$ is the vertex set and $A$ is the edge set. An edge $A_i$ connecting vertices $S_j$ and $S_k$ de-

notes that activity $A_i$ takes the organization from state $S_j$ to state $S_k$. $S_j$ and $S_k$ are known as input and output states, respectively, of the activity $A_i$.

The PAG captures the progression of the organization from a *begin* (input) states to one or more *goal* (output) states, through a sequence of activities and intermediate states, over a period of time.

We now provide an example of how a BP is captured through a PAG, using the ITSU scenario described earlier. Consider an important BP in the ITSU, namely *"Provide User Support"* (PUS), where ITSU users request and receive computing support from the Technicians, with the Lab Monitor acting as a liaison. The PAG for the PUS process is shown in Figure 1. The figure is self-explanatory; however, we provide a brief description of the process below.

1. An ITSU user initiates the process by sending an e-mail/phone support request to the Monitor. This takes the process through states $S_1$ and $S_2$. Clearly, $S_1$ is the *begin* state of process PUS.

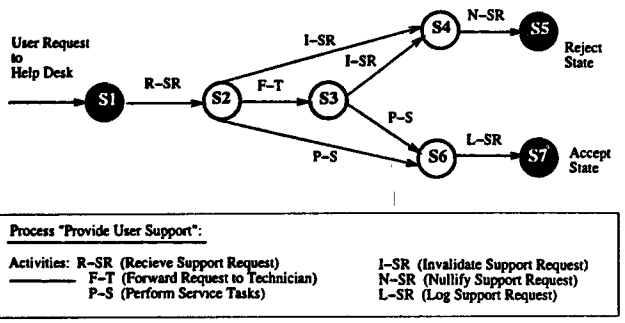
2. If the support request information is complete, the Monitor either forwards the support request to the Technician ($S_3$), or performs service tasks ($S_6$) and logs the support request as "completed" ($S_7$). Else, he/she invalidates the SR ($S_4$), and then cancels or nullifies it ($S_5$).

Thus, $S_5$ and $S_7$ are the two *end* states of this process.

3. Once a Technician receives a support request from the Monitor ($S_3$), he also acts as in step 2.

Figure 1 is a PAG corresponding to the *process model* of the PUS BP. Using such a model one may identify various activity sequences that may be performed to execute instances of a BP. We call such activity sequences *process paths* (analogous to the notion of workflows).

**Figure 1 Process Activity Graph (PAG): Process PUS**




DEFINITION. **Process Path.** A process path is any sequence of activities that connects a "start state—end state" pair in the PAG. Clearly, several process paths are possible for a BP.

The process paths of PUS are as follows: (1) *R-SR, I-SR, N-SR* (2) *R-SR, P-S, L-SR* (3) *R-SR, F-T, I-SR, N-SR,* and (4) *R-SR, F-T, P-S, L-SR.*

## 3. How Well Does the PAG Conform to Comprehensive Business Process Models and Workflows?

In this section we discuss to what degree our PAG construct satisfies the goal of designing a "comprehensive" business process model. To do this, let us first enumerate and examine some desired characteristics of a "good" process model. Let us first ask the question: "what does one do with a model of a business process"? As far as this (and a multitude of other) papers are concerned, a major use of business process (BP) is in *Business Process Reengineering* (BPR) and *Work-Flow Management* (WFM). Both BPR and WFM are concerned with understanding properties of business processes (or work-flows) with the intent of uncovering inherent inefficiencies of these processes/flows. Thus, for a model to be useful to BPR or WFM it must adequately capture properties of the underlying process/flow that it models. If one looks carefully at existing work describing characteristics of business process models (in addition to the references cited previously, see also Abeysinghe and Palp (1997), Manley (1996), and Kueng et al. (1996)) two characteristics emerge that these models must exhibit:

1. Task/activity flow (we shall simply refer to this as activity flow), and

2. Control flow.

Activity flow refers to the task/activity ordering relationships (e.g., input-output, precedence etc.) that is present in a BP. This is also referred to sometimes as information flow (Kamath and Ramamritham 1996), particularly in cases where the activities are transactional, i.e., they can be modeled as transactions operating on some sort of an underlying database. In many ways the activity flow forms the "structure" or "foundation" of a BP model—almost all the definitions of

process models available in the literature (many references cited previously) define processes as activity (or task) flows. In fact, our earlier definition (adapted from Davenport) is no different.

Control flows add additional meaning (or semantics) to the "basic" BP structure offered by task flow models. Many types of control primitives have been proposed in the literature (and as understanding of BP models develop more control structures are being proposed): (a) marking some activities as *vital*, i.e., if these activities failed, the process needs to be aborted, (b) identifying contingency activities, i.e., upon the failure of some (nonvital) activity the BP execution need not be terminated. Rather, some preidentified contingency measures can be adopted, (c) identifying *triggering conditions* for specific activities, i.e, given a conditional branching point certain activities will be triggered based on the occurrence of certain conditions (e.g., the satisfaction of a predicate upon the state of an underlying database). Note the above is just some of the commonly stated control primitives, one can easily conjure up several more. The point is that, unlike activity flow, there exists no "closed form" definition of control flows.

PAGs, obviously, represent the activity flows of a BP model. We are currently working on extending the work presented in this paper to include the discovery of "contingency" and "triggering" conditions—this is elaborated a little more when we talk about future work in the conclusion section. Note however, that the activity flows form the backbone of a process model—without this aspect, there is, in effect, no workable model. Thus, clearly, the PAG represents an important and substantial (though by no means complete) component of a comprehensive model. The logical next question is whether it is indeed possible to specify a complete model. We contend that it is impossible for any model to be complete, because from a control flow standpoint completeness is undefinable—more and more control flow structures are being identified all the time. Indeed, in many respects control flow is application specific, i.e., given a specific application one can identify a set of control primitives that are important. For example, the FEMA (Federal Emergency Management Agency) is in the business of providing assistance to residents of disaster afflicted localities (visit the very informative FEMA web site at http://www.fema.gov). FEMA maintains detailed process models of evacuation plans—clearly, in these models contingency controls take on special meaning as disaster evacuation scenarios tend to be highly unpredictable. This may not be the case in other application processes. In summary, our intuition is that it will be very hard to devise "comprehensive" BP models, and consequently it will be even harder to discover them. Thus, we contend, the PAG represents a critical component of BP models; in fact, the PAG represents a component that *must* be present in all BP models. It has several other advantages as well:

- For simple (e.g., linear or processes with no conditional branchings) processes, the PAG represents most of the information contained in the process
- Given a PAG for a process, a manager, in many cases, would be able to identify much of the control flow in the process. For example if the PAG documented in Figure 1 was shown to the ITSU manager she would immediately be able to deduce that state $S_2$ is a conditional branching point, i.e, from that state only one of the three activities I-SR, F-T, or P-S can be executed for a particular service request. In other words, many control flows can be identified by a careful examination of the PAG by appropriate enterprise personnel. We hasten to add that there may exist complex control events that are not identifiable visually—as mentioned earlier, we are working on systematic extraction of some such control flows.

Basically what we have attempted to show above is that PAGs are a *necessary* component of any BP model. Moreover, in many cases, based on the characteristics of the underlying process, interesting inferences may be drawn about the control flow inherent in the BP being modeled. In addition PAGs have a few more interesting features that aids workflow management (WFM) specifically.

One of the important goals of WFM (Rusinkiewicz and Sheth 1995), is identifying the various workflows that may be executed to accomplish a specific organizational activity, i.e., a business process. It turns out that the output of the PAG model is precisely the above, i.e., a set of process paths as shown in the previous section. Later in this paper, when describing our discovery strategies we state criteria to identify "start"

and "end" states of a process. Essentially, any paths in the PAG between a pair of such states designate a workflow. Having identified such paths, one can subject these workflows to a number of systematic tests to determine whether these are "good". For instance, a path that contains a cycle is possibly inferior than a competing workflow which does not. Note that this is a simple generalization which may not hold in all cases. Moreover, one could potentially record the durations of each task on a PAG (this is easily doable by noting the timestamps of the begin and end events of the task, as noted earlier) and do a comparison between competing process efficiencies.

Thus, we believe that discovering the PAG is a nontrivial problem, the solution to which can add a lot of value to organizations.

## 4. Basic Intuition Behind the Proposed Process Discovery Strategies

We propose to discover a BP by extracting a PAG model of the BP from its *observed behavior*. The first question then is how to observe and record the behavior of a business process. Fortunately, existing process modeling and monitoring literature comes to our aid. It turns out that there exists much work regarding monitoring and recording the behavior of processes (Bradac et al. 1994, Wolf and Rosenblum 1993, Kellner et al. 1990, Cook and Wolf 1995). All of the aforementioned work adopt the view that a process is a sequence of actions. Thus, by recording the observed sequence of actions, an execution trace of the process may be obtained. Actions are characterized by events (i.e., instantaneous, visible occurrences) and consequently, such event data are collected to characterize the process. One may view the collected event data as an *event stream*, which is a sequence of temporally ordered events, i.e., in an event stream event $E_i$ precedes $E_j$, iff

$$\text{Timestamp } (E_i) < \text{Timestamp } (E_j).$$

By fixing appropriate granularities of time, it can be virtually ensured that all recorded events have different timestamps. In other words, event streams can be assumed to be totally temporally ordered. Established procedures exist for collecting such event data (Bradac et al. 1994, Wolf and Rosenblum 1993) and are being used in several industries such as software engineering (in areas such as program visualization, concurrent-system engineering, distributed debugging) and manufacturing. The basic idea behind these procedures is quite simple: observers (or monitors) experience and record the (repeated) enactment of a process by keeping a log of the events that they observe. These events are then organized in temporal order to yield an event stream. Collected over sufficiently long and carefully monitored periods of time, these event streams capture the behavior inherent to a process. To reduce redundancy, as well as keep the length of this paper manageable, we do not state these procedures fully here. Interested readers are referred to the aforementioned papers.

In concordance with existing process modeling literature, we have modeled BPs with events and actions (modeled as *activities* in our case). Thus, we can utilize the same procedures to collect event data. In our case, subsequent to collecting an event stream corresponding to a BP, we convert it to an *activity stream*. This is done very simply as illustrated below.

Consider the following event stream corresponding to the enactment of a business process $P$ with an activity alphabet $\{A, B, C\}$:

$$A^+, B^+, C^+, B^-, B^+, B^-, A^-, C^-.$$

We extract an *activity stream* from this event stream by ordering the activities according to the timestamps of their end events. In other words, for any event stream $E$ and its corresponding activity stream $C$, an activity $A_i$ follows an activity $A_j$ in $C$, iff $A_j^-$ preceded $A_i^-$ in $E$. Thus the activity stream corresponding to the event stream above is

$$B, B, A, C.$$

This activity stream may be regarded as a *trace* of the execution of the BP whose model we are interested in discovering. This trace is nothing but an *example* of the behavior of the BP. The problem now is to design methods to discover a formal model of the process based on this trace. As explained in the previous section, we use a *Process Activity Graph* (PAG) to model a

BP. Thus, we shall design strategies to extract PAGs from activity streams.

### 4.1. Additional Details Regarding Our Approach

Our basic approach to extract process models is to monitor the behavior of a process by observing the activities performed and to record this behavior as an *activity stream*, as described above. More accurately, our approach is to *collect* and *analyze* a trace of process execution and infer a formal model that can account for the organizational behavior encapsulated in the trace. Let us make it clear at this point, that we *do not* expect that this process will always come up with a completely accurate PAG (i.e., process model). For instance, due to inherent, unresolvable properties of our trace collection procedure, we can never guarantee that there is no "noise" in the trace. This is one major difference between software processes inherent to software engineering and human processes inherent to business process engineering. For example, if a software process, $P_s$, includes the activity sequence $A_i$, $A_j$, then it is highly likely that this exact sequence will appear in an execution trace of $P_s$ (unless of course some unlikely events such as program or system errors occur in between). In contrast, human agents are much more unpredictable. For example, if a business process $P_b$ includes the activity sequence $A_k$, $A_l$, it is quite likely that some unprecedented activities may be performed in between e.g., receiving a personal phone call, or going for lunch, or taking a break. The monitoring process, unable to distinguish between "relevant" and "noisy" activities, will record such extraneous activities, which will subsequently appear in the process trace, i.e., the activity stream. Thus, organizational process traces are more likely to be noisier than software process traces. In other words, noise elimination is an important goal of business process discovery strategies. In practice however, complete elimination of noise may be impossible. Thus, it would be difficult to guarantee the accuracy of the PAG generated. However, our objective is to produce a "reasonable" first cut of a BP, which the analyst can then refine. As shown later, our procedures more than fulfill this objective.

As described above, we intend to *collect* process behavioral information in the form of a trace, and then *analyze* this trace to extract a PAG. The focus of this paper is not the data *collection* part. Process monitoring and data collection are well-researched areas, and several methodologies exist (LeBlanc and Robbins 1985, Bradac et al. 1994, Wolf and Rosenblum 1993) that are currently being used in several industries (e.g., software, manufacturing). As a matter of fact, even commercial workflow software (e.g., IBM's workflow product FlowMark (IBM Corporation 1996) supports process monitoring functions. Even better, such software makes it possible to directly record *activity streams* instead of having to first record event streams as would be the case using the methodologies described in the papers cited above.

The focus of this paper is the *analysis* phase, which is conducted once the activity traces have been collected. Our basic strategy is to examine this trace for patterns of repetitive behavior. Recall that the activity stream records activity sequences logged over several execution instances of the process. Thus, if certain patterns recur often enough in the trace, there would be a basis for inferring that these patterns are representative of true process behavior. The logical next question therefore is as follows: "what is a good formal model to represent these patterns?" We found a nice answer by examining the well-known problem of **grammar discovery** (to see a survey of this literature, see Angluin and Smith (1983)). The grammar discovery problem may be stated as follows: *discover the grammar for a regular language, given examples of sentences in that language.* Now, mapping activities to tokens and activity streams (traces) to sentences, the grammar discovery problem maps, to a large degree to the process discovery problem. Based on this analogy, we surmised that similar methodologies may be employed to solve both problems. Now, the grammar discovery problem is a well-studied problem. It turns out that the grammar discovery problem has been approached using *Finite State Machine* (FSM) synthesis, i.e., given examples of its behavior, synthesize (discover) a FSM model that accounts for the behavior. We decided to take a similar tack, i.e., cast the process discovery problem in terms of a FSM synthesis problem. Another advantage of this approach is that the output of this methodology is a FSM which is similar to a PAG, if one considers activities as transitions between states.

Recall from our earlier discussion that our objective is to extract a PAG from process behavior. Thus, the FSM synthesis approach appears to be particularly suitable to fulfill this objective.

Now let us turn our attention to the applicability of existing FSM synthesis methods to our problem. Several techniques have been proposed for discovering FSMs from input-output behavior, most notably by Gill (1962) and Ginsberg (1962, 1966). The problem with applying these directly is that all of these methods require that enough information be included in the problem statement such that the solution is unique. Also these methods do not have the ability to produce unspecified or "don't care" conditions to produce simpler solutions. We need procedures that can handle insufficient information (clearly, regardless of the duration of data collection, one can never guarantee that all pertinent information with respect to a BP has been captured) and yield PAGs (i.e., machines) that often give "reasonable" behavior outside of the specified domain. Also, many of the abovementioned methods need both *positive*, i.e., legal examples of behavior as well as *negative*, i.e., illegal examples of behavior. In our case, obviously, we can only supply examples of actual (i.e., legal) behavior. One method that appears to satisfy some of these criteria is an algorithm postulated by Biermann and Feldman (1972). In particular the strength of this algorithm is that it can take examples of "partial" behavior and produce "reasonable" results. Thus, we start with this algorithm. However, this algorithm (like the others mentioned previously) has two major flaws: (a) the final machine produces too many states, and (b) the procedure is very susceptible to noise. Our goal is to use the Biermann-Feldmann (B-F) algorithm as a starting point and adapt it to the business process engineering framework.

The several FSM synthesis techniques that have been proposed may be classified as *algorithmic* or *probabilistic*. The former used well-understood algorithms to compute grammars from legal and illegal sentences, whereas the latter consists of strategies where sample sentences yield probabilities, which are then manipulated to synthesize FSMs. In keeping with this trend, below, we specify two strategies to extract PAGs from organizational behavior traces, a probabilistic strategy and an algorithmic strategy.

## 5. The Probabilistic Strategy

This strategy was formulated by us and works as follows: it examines the provided activity stream of a BP and uses stochastic process modeling techniques to find the most probable (highest frequency) activity sequences. These probabilities are then algorithmically converted into states and transitions to yield a FSM, which is the PAG model of the BP. Although our algorithm is new, there is other work that has used a similar approach. In Miclet and Quinqueton (1988), the authors use transition probabilities to create FSM recognizers of protein sequences, while in Cook and Wolf (1995), the authors use stochastic techniques for software process modeling.

A *stochastic process* is a mathematical model of a system that varies in time in a random manner. It is defined as any collection of random variables $X(t)$ defined on a common probability space $S$, which is termed as the *state space* of the process.[1] $X(t)$ represents the state of the system at time $t$. The random variables $X(t)$ all take on values from the fixed set $S$.

A *discrete first-order Markov model* is a stochastic process system that has the following properties:

- The process has a finite number of states.
- *The Markov Property: Given the present state of the system, the past states have no influence on the future state.* In other words, at any point in time, the probability of the process being in some state depends *only* upon the previous state of the process.
- The probabilities of the state transitions do not change over time.
- The initial state of the process is defined probabilistically.

In general, an *nth-order Markov model* implies that, at any point in time $t$, the probability of the state $S_t$ of the process being some $X(t)$ depends upon the previous $n$ states the process was in (for an excellent treatment of markov modeling, consult Masaaki (1997)).

We now describe how we extract a model of a BP using stochastic process modeling. More specifically,

[1]Process here refers to the generic definition of processes in the Markov model terminology.

our objective is to generate a PAG characterizing the BP. To start with, a sample activity stream that traces the execution of several instances of the BP is captured, as explained in Section 4. In order to generate the PAG for the BP, an *activity-sequence probability matrix* is constructed from the activity stream in the following way: traversing the sample activity stream, and, for each activity sequence, tallying the occurrences of all possible future activities one by one in the form of a matrix.

Now, the probabilistic strategy, using an $n$th order Markov model, proceeds in the following steps. Note that while at first reading, the description may sound cryptic, the subsequent example should clarify the description.

1. Construct the nth order activity-sequence probability matrix, given the sample activity stream, in the following manner:

(a) Traverse the sample activity stream. For each activity sequence $a_i$ of length $n$ (where $n$ is the order of the model), tally the number of occurrences of all possible future activities.

(b) Calculate the probabilities of each of these future activities occurring after the activity sequence $a_i$, by dividing the tally for the future activity, by the total number of occurrences of $a_i$ in the activity stream. Represent the activity probabilities for each activity sequence in the activity stream in the form of a *probability matrix*.

2. A *threshold probability* is fixed, as a parameter. A directed graph called the *activity graph* (*AG*) is constructed from the probability matrix, in the following way:

(a) Assign a node to each activity type in the activity stream.

(b) Then, pick out all the activity sequence-future combinations, of length $n + 1$, that exceed the threshold probability. For each of these sequences, create uniquely labeled edges from each element (node) in the sequence to its immediately succeeding element (node) in the same sequence, for all the nodes except the last node.

3. The activity graph *AG* created in the previous step (possibly) includes repetitive (duplicate) edges between the same pairs of nodes. These are eliminated, such that, if any node-node pair in the *EG* is connected by one or more edges, then all but one of these edges are removed, and only one connecting edge is retained. Then, in the resultant *AG*, all the remaining edges are newly and uniquely labeled. This step ensures that, in the final PAG created in the last step, redundant states are merged with the retained states. More specifically, if the same transition path $A_i$ leads from state $S_i$ to states $\{S_1, S_2, \ldots, S_n\}$, and all these destination states have the same output transition path(s), then all the states $\{S_1, \ldots, S_n\}$ are merged into one common state $S_j$. Then, the occurrence of transition path at state $S_i$ takes the organization to the state $S_j$ only.

4. The graph *AG* is converted into its *dual*, *AG'*, using the following steps:

(a) Each edge in *AG* becomes a node in *AG'*.

(b) For each in-edge/out-edge pair of a node in *AG*, we create an edge in *AG'* from the node corresponding to the in-edge of the pair to the node corresponding to the out-edge of the pair.

(c) The edge thus created is labeled by the activity node in the *AG* for this pair of edges.

These steps result in a *coarse* PAG (*AG'*), which still may have *illegal* activity sequences (whose probability is less than the threshold level). Examples of this are shown below.

5. Lastly, *AG'* from the above step is converted into the final Process Activity Graph for the BP in consideration, in the following manner: All the illegal activity sequences are removed from *AG'*, by removing those edges in *AG'* that cause illegal sequences, and at the same time, taking care that, in doing so, none of the legal sequences are removed.

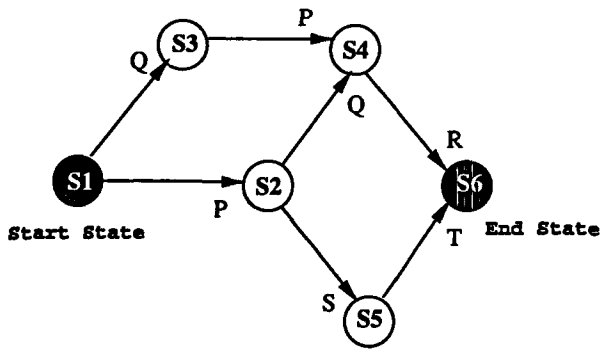
The graph constructed in this last step is the final Process Activity Graph (PAG) for the BP in consideration. Note that as a result of step 5 we cannot guarantee that all illegal sequences will be removed from the PAG.

EXAMPLE. As an illustration, consider the ITSU organization from our example scenario, and a typical BP *"Reserve Equipment"* (ResEqp). This process is performed by the Lab Monitor, and involves the loaning of computer equipment to the ITSU users. Upon an equipment request (typically by phone or e-mail), the Lab Monitor checks an equipment-reservation database. If the equipment has no previous reservations, he/she makes the new reservation. Otherwise, the request is added to a waitlist, and then serviced upon

availability. The five activities of the ResEqp process are: "Get-Equipment-Request", "Check-Equipment-Available", "Reserve-Equipment", "Check-Equipment-Unavailable", and "Waitlist-Equipment-Request". For brevity, we shall refer to these activities as "*P*", "*Q*", "*R*", "*S*", and "*T*" respectively. Thus, the activity alphabet $\Sigma_{ResEqp}$ = [*P*, *Q*, *R*, *S*, *T*]. It is easily seen from the above description that the ResEqp process always begins with a user request for equipment reservation, and, depending upon the availability of the equipment, the process ends either with the "Reserve-Equipment" activity or with the "Waitlist-Request" activity. In other words, the ResEqp process always starts with the activity *P*, and culminates in either *R* or *T*. Note that since a waitlist exists, it is possible that activity *P* draws its input either from a direct user request, or from the waitlist. In the latter case, it is possible that activity *Q* precedes *P*.

The reason for choosing the ResEqp process, is that it is intuitively very easy to understand and model. In other words, this process is simple enough and understood well enough, such that we could come up with a "correct" and unambiguous model for it. By examining the activities involved, we can come up with a very good complete PAG for the ResEqp process, as shown in Figure 2. This enables us to evaluate how good our strategies are by comparing their output to this model. The PAG in Figure 2 shows all the possible process paths of the ResEqp process. These are the different paths between the start state (S1) and and end state (S6). They are: *P-Q-R*, *P-S-T*, and *Q-P-R*. A good model of this process would yield these process paths.

**Figure 2** **PAG for the ResEqp process**




Let us now see how the model generated with the probabilistic strategy performs.

The first step was to capture a trace of this BP by collecting an activity sequence representing repeated enactment of this process. This is shown below:

PQRPQRPSTPQPRQPRQRPSTQPRPSPSTT

Next, we describe the process of generating a PAG for this BP using the probabilistic strategy. We employ a second-order probability matrix in this example.

1. The activity-sequence probability matrix is constructed for the sample activity stream, as shown in Table 1. We consider every 2-activity sequence in the activity-stream, and, for each future activity possible, we tally its actual number of occurrences after the 2-activity sequence. Then the probabilities of each future activity occurring after each 2-activity sequence are calculated and tabulated to form the matrix, as in Table 1. (The probability $p$ of any future $f_i$ occurring after an activity sequence history $h_i$ is given by the ratio of number of occurrences of $f_i$ succeeding $h_i$, to the total number of occurrences of $h_i$ in the activity stream). For the sample activity stream earlier in this section, we can see from Table 1 that the activity-sequence *Q-R* is always followed by a *P*, and never by a *Q* or *R*. Similarly, *P-S* is followed by *T* three out of every four times, and so on.

Note that construction of this matrix is an illustration of the exploitation of *n*th order markov properties alluded to earlier. The matrix shows the probability

**Table 1** **Probabilistic Strategy: Second-Order Activity Probabilities**

| Act. | P | Q | R | S | T |
|---|---|---|---|---|---|
| *PQ* | 0.33 | 0.00 | 0.67 | 0.00 | 0.00 |
| *QR* | 1.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| *RP* | 0.00 | 0.25 | 0.00 | 0.75 | 0.00 |
| *PS* | 0.25 | 0.00 | 0.00 | 0.00 | 0.75 |
| *ST* | 0.34 | 0.33 | 0.00 | 0.00 | 0.33 |
| *TP* | 0.00 | 1.00 | 0.00 | 0.00 | 0.00 |
| *QP* | 0.00 | 0.00 | 1.00 | 0.00 | 0.00 |
| *RQ* | 0.50 | 0.00 | 0.50 | 0.00 | 0.00 |
| *PR* | 0.33 | 0.67 | 0.00 | 0.00 | 0.00 |
| *TQ* | 1.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| *SP* | 0.00 | 0.00 | 0.00 | 1.00 | 0.00 |

that a specific activity will occur following a history whose length is given by $n$. The underlying assumption of course is that knowing a history is enough to "completely" characterize the possible futures. For example, given the history $PQ$ (i.e., knowing that the last activity performed was $Q$ and it was immediately preceded by $P$), from Table 1, we will claim that the only possible future activities are $P$ (with a probability of $\frac{1}{3}$) and $Q$ (with a probability of $\frac{2}{3}$). In other words, the futures exclusively depend upon a history of length $n$ (i.e., $n$ state transitions, where the $i$th transition is simply the occurrence of the $i$th activity in the sequence, $i \leq n$). This is nothing but the exploitation of the Markov property. Finally we comment on the choice of $n$. Essentially, $n$ is the length of the history that we choose to characterize futures. It is trivial to see therefore that $n$ can take on any discrete value in the interval $[1, |\Sigma|]$, where $\Sigma$ is the activity alphabet, i.e., the set of possible activities that characterize the BP under observation. The choice of $n$ is up to the analyst. Clearly in this example $n$ is chosen to be 2. The larger the $n$, the more fine-grained will be the resulting matrix. However this fineness will come at the cost of (potential) additional complexity to the system. In the example shown $n$ was chosen to be 5, and it is easily seen that the matrix would have had 120 rows (in general the number of possible histories, i.e., the number of rows in the activity-sequence probability matrix, is given by the number of permutations of $n$ elements from a set of $|\Sigma|$, i.e,

$$^{|\Sigma|}P_n = \frac{|\Sigma|!}{(|\Sigma| - n)!}).$$

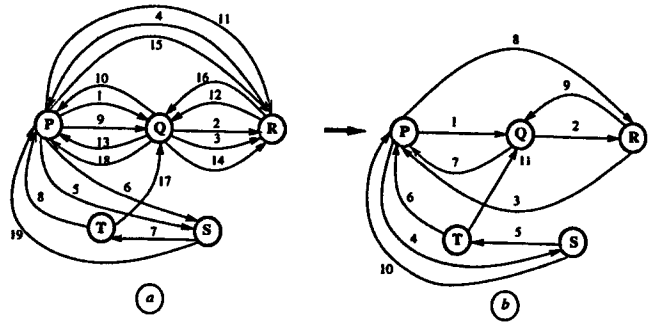
A very small $n$ on the other hand would possibly hide a lot of system internals and yield an overly simplistic model. Our sense is that the designer would experiment with a number of different $n$ values and choose the model that appears to capture reality the best.

2. A *threshold probability* is fixed, as a parameter. In this example, we fix the threshold probability as 0.50. The *activity graph* (*AG*), as shown in Figure 3-a, is constructed from the probability matrix, using the following steps:

(a) The vertices $P$, $Q$, $R$, $S$, and $T$ are created, representing each activity type.

(b) All the 2-activity sequences and corresponding

**Figure 3 Probabilistic Strategy: Activity Graph (*AG*)**




future activities, that exceed the threshold probability, are selected, from Table 1. Thus, the sequences *P-Q-R*, *R-P-S*, *P-S-T*, *Q-R-P*, *T-P-Q*, *Q-P-R*, *R-Q-P*, *R-Q-R*, *P-R-Q*, *T-Q-P*, and *S-P-S* are selected. These are the *legal* sequences. For each of these sequences, we create uniquely labeled edges from each node in the sequence to the immediately succeeding node in the same sequence. For example, for the activity sequence *P-Q-R*, whose probability is 0.67 (from the first row of Table 1), we create edge 1 from node $P$ to node $Q$, and edge 2 from node $Q$ to node $R$. This process is repeated for all the legal sequences. Thus, the edges 1 to 19 are created, as in Figure 3a.

3. All the repetitive (duplicate) edges between the same pairs of nodes in *AG* are removed, and the resultant *AG* is created, as shown in Figure 3b, with nodes $P$, $Q$, $R$, $S$, and $T$, and uniquely labeled edges 1 to 11.

4. The graph *AG* is converted into its *dual*, *AG'* (shown in Figure 4), as follows:

(a) Nodes labeled $S_1$ to $S_{11}$ are created, corresponding to each edge in *AG*.

(b) Edges are created in *AG'* as follows: for each in-edge/out-edge pair of a node in *AG*, we create an edge in *AG'* from the node corresponding to the in-edge of the pair to the node corresponding to the out-edge of the pair. For example, consider the edges 1 and 2, leading into and out of node Q, as shown in Figure 3b. In the dual graph, i.e., *AG'*, shown in Figure 4, create an edge from node $S_1$ to node $S_2$, and label it "*Q*". Similarly, we create an edge labeled "*R*" from node 2 to node 3, and so on.

5. Lastly, *AG'* from the above step is converted into the final Process Activity Graph for the example BP by

removing all the edges in $AG'$ that cause illegal sequences. (The legal sequences are listed in Figure 5a).

The resultant PAG is shown in Figure 5b. For example, the edge $Q$, connecting nodes 1 and 7 in $AG'$, are removed, thus eliminating the illegal sequence $P$-$Q$-$P$. The edges $S$ connecting nodes 4 and 10, and $P$ connecting 10-8 and 10-1, are removed to eliminate the sequence $P$-$S$-$P$. This step is carried out for all illegal sequences occurring in $AG'$ (i.e., all sequences whose probability is less than the threshold probability). The resultant PAG is created, as shown in Figure 5. Each of the nodes represent a state of the process, and are therefore relabeled as $S_1, \ldots, S_{11}$.

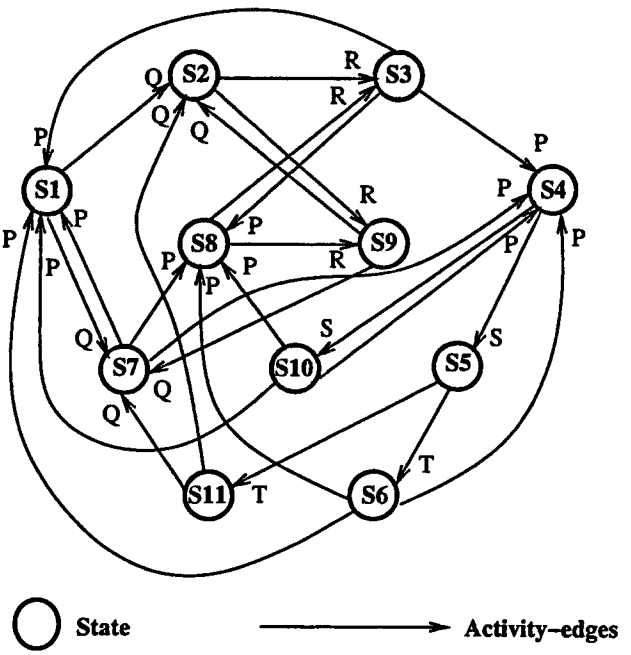
The reader may be curious as to how we determined the start and end states in Figure 5b. In particular it is indicated in this figure that the possible start states are $S_3$, $S_7$, $S_{10}$ and $S_{11}$, while the possible end states are $S_3$, $S_6$, $S_9$ and $S_{10}$. Our rationale for this determination is provided in the description of the ResEqp process at the beginning of the current example. Recall that the ResEqp process is always initiated with the activity "$P$" and may be terminated with the activities "$R$" or "$T$". Thus any state having an output transition of "$P$" is a possible start state and any state having input transitions of "$R$" or "$T$" is a possible end state.

In general, the PAG created by the probabilistic strategy will possibly include some *nondeterministic* transitions, but it can be reduced to a deterministic FSM model using techniques described in Martinsons (1995). The robustness of this model can be controlled through the *threshold probability* parameter. In the next section, we describe our purely algorithmic process-modeling strategy. Note that in this section we did not comment on the *quality* of the model produced by the probabilistic strategy, i.e, the PAG of Figure 5b. We postpone this discussion until we have shown the models generated by our other algorithms. Finally, a comprehensive commentary regarding these models is provided in Section 7.

## 6. The Algorithmic Strategy

In this section, we describe a number of algorithmic process modeling strategies. These strategies are based on the Biermann-Feldman algorithmic model for synthesizing machines from finite subsets of their input-output behavior (Biermann and Feldman 1972).

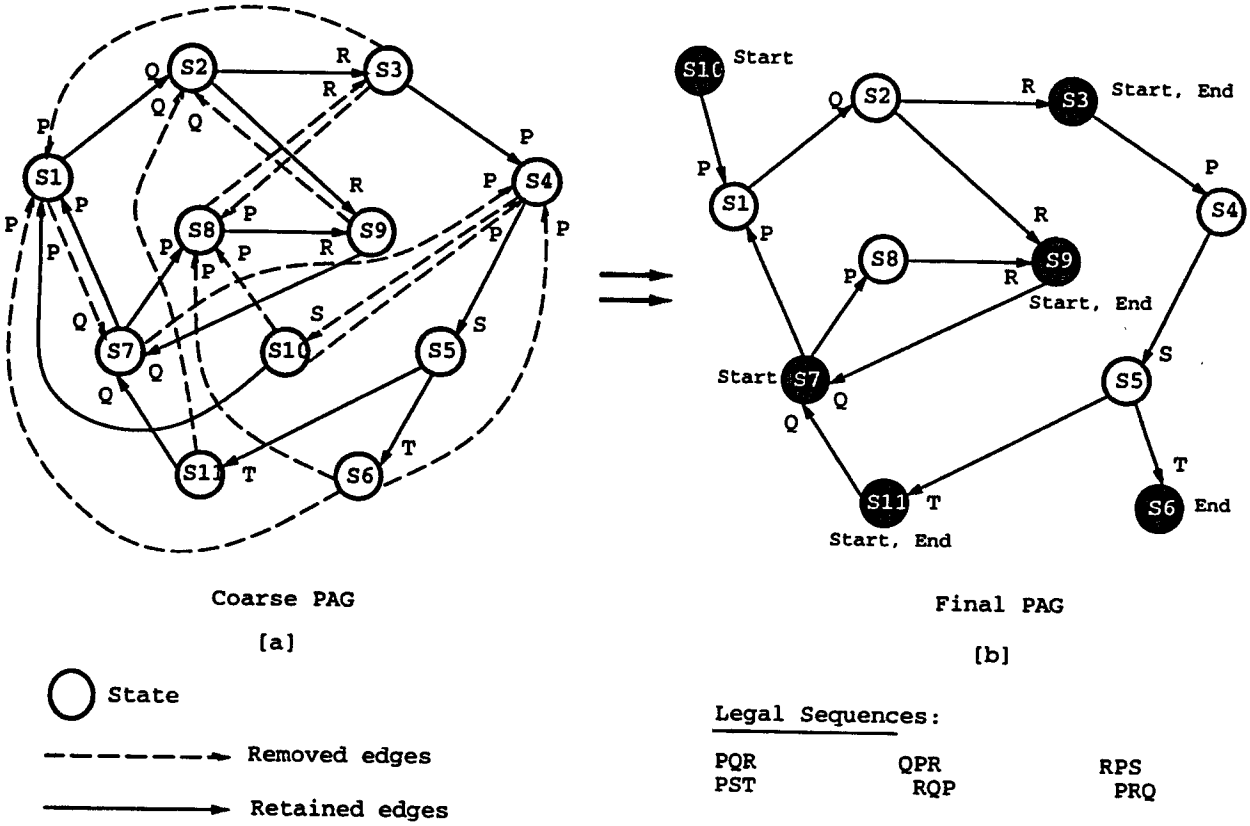
**Figure 4 Probabilistic Strategy: Dual Activity Graph (*AG'*)**




Our objective here is to extract a model of a BP algorithmically, from a sample activity stream collected during the process execution (as in the probabilistic strategy). More specifically, our goal is to generate the PAG from systematically collected process data, described in Section 4.

The original B-F algorithm, described in Biermann and Feldman (1972), is a modification of the Nerode realization technique (Nerode 1958) for synthesizing *finite state machines* (FSMs) that characterize or realize some finite-state computable function $f$. The algorithm is concerned with designing a FSM for a given machine or system, from only a finite number of its behavior samples, when no other formal representations of the machine are available. The resultant FSM has output values associated with each state; however, in our algorithmic strategy, we adapt the algorithm to generate a PAG for an organization, given sample activity streams characterizing a BP. The central idea behind the original B-F algorithm is as follows: *Any state of a process is defined by what future behaviors can occur from it*. In a sample activity stream collected on the process, for a given *history* (continuous sequence) of activities, the current state reached by that history is determined

**Figure 5** **Probabilistic Strategy: Final PAG**




by what future sets of activities can occur from it. The length of the "future" sequence of activities is set as a parameter $k$ to the algorithm. Therefore, we refer to the original algorithm as B-F($k$). If two different histories result in the same future sequence of activities, then they are said to reside in the same *equivalence class.* Each equivalence class thus derived, represents a state in the final FSM. The FSM represents the PAG for the process. We now formally describe the B-F($k$) algorithm.

### 6.1. B-F($k$) Algorithm

Let $S$ be an activity stream collected by observing the repeated enactment of a BP which we wish to model. Let $\Sigma = \{A_1, A_2, \ldots, A_m\}$ be the activity alphabet that characterizes the BP being modeled. Let $H$ be the set of all activity sequences possible in $S$, including the full activity stream in $S$. Then $h \in H$ represents any valid activity sequence. The activity sequences $h \in H$ are henceforth referred to as *histories.* Let $k$ represent the allowable length of future activity sequences, as a parameter. Now we define the set $F$, as the set of all activity sequences $f_i$ composed from $\Sigma$, of length $k$. The activity sequences in $K$ are referred to as *futures.* Then $f \in F$ represents any valid future.

The objective is to define a set of equivalence classes $C = \{C_1, \ldots, C_s\}$, which will represent the states of the resultant PAG for the process.

DEFINITION. **Equivalence Classes.** An equivalence class $C_i$ is a set of histories $h \in H$ such that, all such histories have an identical future $f \in F$. More specifically, if two histories $h$ and $h'$ have the same future $f$ (of length $k$), then $h$ and $h'$ belong to the same equivalence class. It is possible that an equivalence class can be comprised of a single history. The whole set of histories $h \in H$ is classified into a set $C = \{C_1, \ldots, C_s\}$ of equivalence classes. The equivalence classes in $C$ are mapped to states in the PAG.

The parameter $k$ controls the complexity and determinism of the generated model. If $k$ is small then there will be a large number of possible futures, i.e., large number of states in the model, leading to potentially a high degree of nondeterminism. One the other hand, a large $k$ will reduce the number of states, perhaps, in the process, sacrificing the observance of subtleties of the underlying process, but reducing the complexity.

The B-F($k$) algorithm creates an PAG using the following steps:

1. All the equivalence classes $C_1, \ldots, C_s$ are extracted for the given sample activity streams that characterize the process. Each equivalence class $C_j$ is represented as a unique state $S_j$ of the process. A graph is created with nodes $S_1, \ldots, S_s$.

2. For each $C_j$ (input state), transitions to *destination states* $C_k$ upon each activity $A_i$ are determined as follows:

(a) For each history $h$ in $C_j$, create a concatenated activity sequence $q$, where $q = h.A_i$.

(b) Now, if $q$ is found to belong to some other equivalence class $C_k$ (i.e., if $q \in C_k$), then $C_k$ is termed as a destination state of activity $A_i$ from state $C_j$.

(c) An edge is created from state $S_j$ to each destination state $S_k$ corresponding to $C_k$, and it is labeled as "$A_i$".

(d) If no such destination state can be found, then no transition exists out of state $C_j$ upon the occurrence of activity $A_i$. (It is also possible that there are more than one transitions from an input state upon an activity).

3. The resultant graph is the final PAG for the process. Each of the states $S_j$ in the PAG is mapped to a set of histories in $H$.

### 6.2. B-F($k$) Example

To exemplify the B-F($k$) algorithm, we now consider the same example activity stream AS, as shown in Section 5, characterizing our ITSU organizational scenario described in Section 2. For this example, we fix the value of $k$ as 3. In other words, each future activity sequence $f$ is made up of 3 successively occurring activities. The activity alphabet is $\Sigma = \{P, Q, R, S, T\}$. Some of the 3-activity futures actually occurring in AS are as follows: *PQR, QRP, RPQ, RPS, PST,* and so on. In all, there are 19 unique 3-activity futures occurring in AS, and therefore, 18 equivalence classes. The set $H$ of all possible histories is created as follows:

$$H = \{P, PQ, PQR, PQRP, \ldots, PQRP, \ldots, PSPSTT\}.$$

Basically, $H$ includes all possible *prefixes* (to any future in AS). Now the B-F($k$) algorithm proceeds as follows:

1. All the equivalence classes $C_1, \ldots, C_{19}$ are extracted for the given sample activity streams that characterize the process. Each equivalence class $C_j$ is represented as a unique state $S_j$ of the process. A graph is created with nodes $S_1, \ldots, S_{19}$, each representing an equivalence class for the corresponding future. This graph, shown in Figure 6a, is referred to as the PAG for the process. The state $S_1$, containing all histories that lead to the future PQR, is as follows: $S_1 = \{PQR\}$, State $S_3 = \{PQ\}$. Similarly, histories are grouped under each of the other states.

2. For each $S_j$ (input state), transitions to *destination states* $S_k$ upon any activity $A_i \in \{P, Q, R\}$ are determined by the following steps. For example, consider state $S_1$.

(a) For each history $h_i \in S_1$, create a concatenated activity sequence $q$, where $q = h.A_i$. For example, let us consider the occurrence of activity $P$. Then, when the history in $S_1$ is concatenated with $P$, we see that the resultant history $q$ is as follows: $\{PQRP\}$.

(b) Now, we find that $q$ belongs to the equivalence class $S_2$, whose future is "$QRP$". Therefore, state $S_2$ is a destination state of activity $P$, from state $S_1$.

(c) A transition edge is created from state $S_1$ to destination state $S_2$, and it is labeled as "$P$", as shown in Figure 6a.

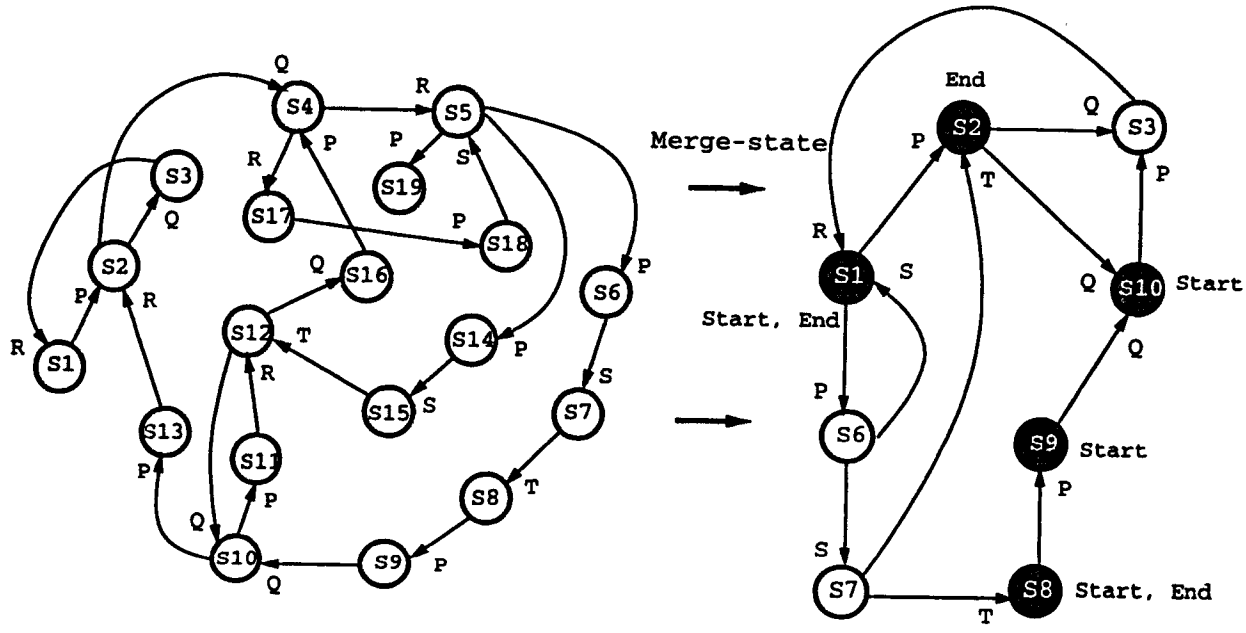
(d) The steps a, b, and c are repeated for every history in the state.

3. Step 2 is repeated for every state $S_1$ to $S_{19}$. For example, state $S_2$, by the above steps, has a transition edge to state $S_3$ and $S_4$ upon activity "Q", state $S_{13}$ has a transition edge to state $S_2$ upon activity "$R$", and so on.

4. If no such destination state can be found, then no transition exists out of state $S_j$ upon the occurrence of activity $A_i$. In our example in Figure 6, however, all states have output transitions. Also, there are more than one transition from some states: state $S_2$, upon activity Q, $S_4$, upon $R$, etc.

5. The resultant graph in Figure 6a is the final PAG for the process.

**Figure 6 PAG: B-F(*k*) Strategy**




The PAG produced by the B-F(*k*) algorithm is always correct, as opposed to the probabilistic strategy in Section 5. But it may be overly complicated due to the creation of more than one destination states for the same transition path which have the same output transitions. In order to minimize the number of states and thus simplify the PAG, we propose the *merge-state* technique. (A similar technique was described in Cook and Wolf (1995)). For each transition path (activity) $A_i$, the merge-state technique proceeds as follows:

1. If a state $S_j$ has transitions to destination states $S_k, \ldots, S_n$ upon activity $A_i$, and if the set of output transition activities for the states $S_k, \ldots, S_n$ are equivalent, then the states $S_k, \ldots, S_n$ are merged into one unique state $S_k$.

2. Now, an output transition path is created from the state $S_j$ to the state $S_k$, and labeled by the activity $A_i$.

For example, in the PAG in Figure 6a, state $S_2$ has transitions to both state $S_3$ as well as $S_4$, upon activity Q. Further, both $S_3$ and $S_4$ have the same output transition activity R. Therefore, states $S_3$ and $S_4$ are merged, and the new state is renamed $S_3$. Now, $S_3$ has transition edges labeled "*R*" to $S_1$ and $S_5$, and both have the same output transition edge "*P*". Therefore, states $S_1$ and $S_5$ are merged, and the new state renamed $S_1$. The resultant PAG, after applying the merge-state technique, has only eight states, as shown in Figure 6b. But, even after minimizing the number of states in the PAG as in Cook and Wolf (1995) the resultant PAG still lacks robustness in the presence of *noise* in the activity stream. (The notion of noise is clarified in the following section). In order to counter the noise problem and infuse robustness into the model, we propose a modified B-F algorithm to create an organizational PAG, that includes additional input parameters for robustness. In the following section, we describe our proposed B-F(*k*, *c*) strategy. As in the previous section, we defer a discussion of the "goodness" of the model shown in Figure 6b until Section 7.

### 6.3. The B-F($k$, $c$) Strategy

The PAG produced by the B-F($k$) algorithm, as in Figure 6, may still not represent an entirely correct PAG for the organization. This is because, the activity stream trace, collected by human agents, is likely to have *noise*, defined below.

DEFINITION: **Noise** $A$. "Noise" activity is any unrelated extraneous activity, executed by a participating agent, which is not actually a part of any activity in the process.

For example, in the ITSU scenario, collector-recorded activities such as "Received phone call for a friend", or "Started lunch break" may not be significant to the *process* that is being discovered from the activity stream. More specifically, let activities $A_1, \ldots, A_4$ occurring continuously in the activity stream represent an activity. Now if a noise activity $A_x$ occurs between activities $A_2$ and $A_3$, it causes the activity sequence ($A_3$, $A_4$) *not* to be regarded as a future for the history ($A_1$, $A_2$). In other words, the noise activity $A_x$ disrupts a possibly continuous stream.

Some noise activities have to be expected and eliminated as the process discovery strategy proceeds. This is more important in case of the algorithmic strategy, where probabilities are not taken into account. To counter this noise problem, and introduce some measure of *confidence* into the resulting PAG, we propose a modified approach, termed as the B-F($k$, $c$) strategy. The B-F($k$, $c$) strategy proceeds in the following steps:

1. The first step of the strategy proceeds similarly as step 1 of the B-F($k$) algorithm. It involves the creation of the *equivalence classes* $C_1, \ldots, C_s$, from the given activity stream AS, with futures of length $k$. As these equivalence classes will undergo further pruning as this strategy progresses, we refer to them, at this stage, as *Rough Equivalence Classes (REC)*.

2. Now, we introduce a new parameter $c$ into the strategy, where $c$ is the *confidence factor* we apply, in order to infuse robustness and minimize the noise problem.

DEFINITION. **Confidence Factor** $c$. The confidence factor is a frequency parameter that selects, from an equivalence class $C_i$ of future $f_i$, only those histories $h_i$ that immediately precede $f_i$ with a frequency greater than or equal to $c$ in $AS$.

In other words, we feel *confident* that a history $h_i$ indeed leads to a future $f_i$, if its confidence factor is $\geq c$. Typically, $c$ values of 0.9 or so are chosen, which simply means that 90% of the occurrences of $h_i$ in the activity stream are immediately followed by the future $f_i$. Therefore, we can confidently choose $h_i$ as a valid history to $f_i$. Now the strategy proceeds as follows:

(a) Fix the $c$ value as desired.

(b) Let each Rough Equivalent Class (REC) $C_i$ be defined as leading to the future $f_i$. Then, for each $C_i$, retain a history $h_i$ in $C_i$ only if the ratio of the *number of occurrences of $h_i$ followed by $f_i$*, to the *total number of occurrences of $h_i$ in the activity stream AS*, is greater than the chosen confidence factor $c$.

(c) Retaining only those histories in the RCS whose confidence is $\geq c$ results in the formation of *Final Equivalence Classes (FEC)*. After the introduction of the confidence factor $c$ in our strategy, we redefine final equivalence classes, as follows:

DEFINITION. **Final Equivalence Classes.** A final equivalence class $C_i$ is a set of histories $h \in H$ such that, all such histories have an identical future $f \in F$, and a confidence factor $\geq c$. More specifically, if two histories $h$ and $h'$ have the same future $f$ (of length $k$), and each have a confidence factor $\geq c$, then $h$ and $h'$ belong to the same equivalence class. It is possible that an equivalence class can be comprised of a single history. The whole set of histories $h \in H$ is classified into a set $C = \{C_1, \ldots, C_s\}$ of equivalence classes. The equivalence classes in $C$ are mapped to states $S_1, \ldots, S_s$ in the PAG.

3. Now the rest of the strategy proceeds as in the B-F($k$) strategy, steps 2 and 3.

4. The resultant graph is nothing but the PAG for the process, with nodes representing states, and edges representing transition activities.

It can be seen that, in a single sample activity stream, it is not possible for histories to have more than one future, as each history, being a prefix, is unique. Therefore, in order to apply the B-F($k$, $c$) strategy to an organization and generate the PAG, we need to consider more than one sample activity streams captured as the organizational process iterates.

EXAMPLE. To exemplify the strategy, let us consider set $S$ of the following three example activity streams $AS_1$, $AS_2$, and $AS_3$ characterizing our example ITSU

scenario described in Section 2, and the BP "Reserve Equipment" described in Section 5. The first activity stream $AS_1$ is almost identical to the one that was used in Section 5.

$AS_1$: *PQRPQRPSTPQPRQPRQRPSTQPRPSTPSP*
$AS_2$: *PQRPQRPSTPQRQPRQPRPSTQPRPSTPSP*
$AS_3$: *PQRPQRPSTPQRQPRQPRPSTQPRPSTPST*

The activity alphabet is $\Sigma = \{P, Q, R, S, T\}$. To apply the B-F($k$, $c$) strategy to these activity streams, we consider the set $H$ of histories $h_i$ such that each $h_i$ belongs to at least one, and possibly more than one of the three activity streams. For this example, let us fix $k = 3$ (activities), and $c = 0.6$. We can see that 19 3-activity futures are possible, out of $AS_1$, $AS_2$, and $AS_3$. Now the *B-F(k, c) strategy* proceeds as follows:

1. A graph is created with nodes $S_1, \ldots, S_{19}$, each representing a rough equivalence class (REC) for these futures. It is important to note here that histories for the corresponding future are to be extracted from all three activity streams.

2. Now, for each of the RECs $S_i$, we eliminate those histories that do not satisfy the confidence factor of 0.6. For example, the history *PQRPQRPSTPQ* is removed from the REC $S_{10}$, as it has a confidence factor of only 0.33. In other words, this history occurs thrice in $S$, but is followed by the future PRQ only once, giving its confidence factor for PRQ as $1/3 = 0.33$. It remains in the REC $S_{11}$, as its confidence factor for future RQP is 0.67. This ensures that this history resides in only that REC whose future it will most often lead to.

3. As in the above step, all the RECs are converted into final equivalence classes (FEC).

4. For each state $S_i$, transition edges to other states, upon each of the activities $P$, $Q$, $R$, $S$, and $T$ are created in the graph, as explained in the B-F($k$) algorithm in Section 6.1. For example, the concatenation $q$, of one of the histories in state $S_1$ with activity $P$, belongs to the FEC $S_2$. Therefore an edge labeled $P$ is created between $S_1$ and $S_2$. Similarly, an edge labeled $S$ is created from $S_6$ to $S_7$, and so on. No transitions exist into or out of states $S_8$ and $S_9$. This means that, the histories in these states are very rare, and represent "noisy" and non-characteristic activity combinations. Therefore states $S_8$ and $S_9$ are not significant to the final PAG. The resultant graph is shown in Figure 7a.

5. Now the merge-step is applied to the above graph. The states $S_3$ and $S_4$, $S_1$ and $S_5$, $S_6$ and $S_{14}$, $S_7$ and $S_{19}$, $S_{10}$ and $S_{16}$ are merged, initially. Then the states $S_1$ and $S_5$, $S_7$ and $S_{13}$, $S_3$ and $S_{11}$, and $S_2$ and $S_{12}$ are merged, resulting in a total of 7 states in the final graph. The resultant graph, shown in Figure 7b, is the final PAG for the organization characterized by the activity streams in $S$.

In the following section, we discuss the relative merits and demerits of each of the three strategies so far described.

## 7. Discussion of the Strategies

We present a discussion of the results of the three process discovery strategies described in the two previous sections. In this discussion we will try to comment upon how "good" the strategies are, both individually as well as comparatively. We use three metrics to evaluate the strategies:

(a) *The number of "correct" process paths produced:* recall that the correct process paths for the process under consideration were described in Section 5. In particular these paths were *P-Q-R*, *P-S-T* and *Q-P-R*. This metric reveals the ability of a particular strategy to recognize repeating behavioral patterns.

(b) *The number of "incorrect" or spurious process generated:* this metric yields insight into how susceptible a strategy is to noise.

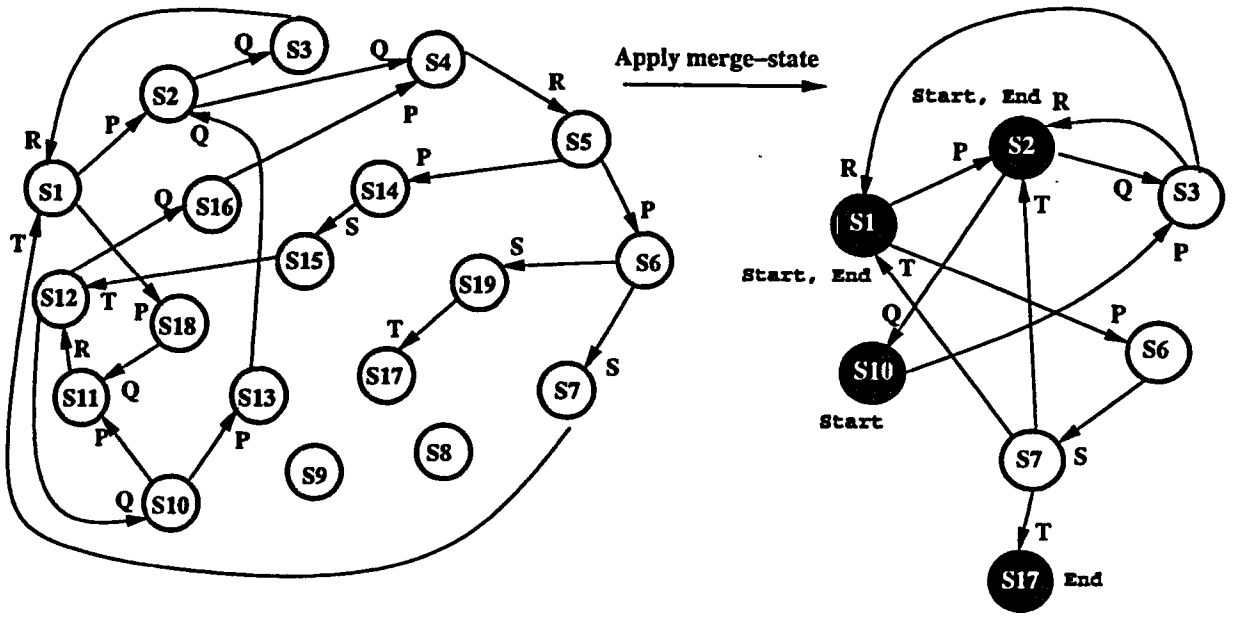
(c) *Number of states that exist in the final PAG:* this metric captures the complexity of the final model. The ideal strategy is clearly one which would yield all the correct process paths, without any spurious ones, with as few states as possible.

**1. The Probabilistic Model.** The model output by this strategy is shown in Figure 5b. We make the following observations regarding this model.

1. States $S_3$, $S_7$, $S_{10}$ and $S_{11}$ are the potential start states. $S_3$, $S_6$, $S_9$, and $S_{11}$ are the end states. In all this strategy produces 11 states, the most by any strategy. Thus the complexity of this model is the greatest.

2. By tracing paths connecting start and end state pairs, we can see that the PAG captures all three correct process paths: (a) *P-Q-R* (linking start state $S_{10}$ to end

**Figure 7** PAG: B-F(*k*, *c*) Strategy




**Initial PAG**

**[a]**

**Final PAG**

**[b]**

state $S_3$), (b) *P-S-T* (linking start state $S_3$ to end state $S_6$), and (c) *Q-P-R* (linking start state $S_{11}$ to end state $S_9$). In other words, the probabilistic strategy is successful at recognizing correct process behavior. The reader can also verify that several other possible process paths in the figure (such as those connecting $S_7$ to $S_3$ or $S_3$ to $S_{11}$) also belong to the correct set.

3. It can be seen that the probabilistic strategy captures the spurious path *P-R* (linking start state $S_7$ to end state $S_9$), which is semantically incorrect; "Get-Equipment-Request" (*P*) cannot lead to "Reserve-Equipment" (*R*) directly, without the occurrence of "Check-Equipment-Available" (*Q*) in between *P* and *R*. This happens mainly because the single sample activity stream used may contain "noisy" sequences.

**2. The B-F(*k*) Strategy.** The model produced by this strategy is shown in Figure 6b. The following observations may be made about the final PAG resulting from the B-F(*k*) strategy.

1. This strategy yields a less complex final model than the probabilistic strategy, containing 8 states.

2. This strategy yields 2 of the three process paths: (a) *P-Q-R* (linking start state $S_1$ to end state $S_1$), and (b) *P-S-T* (linking start state $S_1$ to end state $S_8$). It fails to capture the *Q-P-R* sequence.

3. This strategy captures two spurious paths, *P-R* (linking start state $S_{10}$ to end state $S_1$) and *P-S* (linking start state $S_1$ to end state $S_1$, through the intermediate state of $S_6$). Why *P-R* is incorrect is explained above in the discussion for the probabilistic strategy. According to the description of the ResEqp process in Section 5, *P-S* makes very little sense; the activity sequence Get-Equipment-Request" and "Check-Equipment-Unavailable" cannot comprise a process path in itself. It needs to be terminated by the activity *T* ("Waitlist-Equipment-Request"). It is interesting to note that in the probabilistic strategy, process path *P-S* had been avoided due to the threshold probability factor. Also,

it appears that the B-F($k$) strategy is more susceptible to noise than the probabilistic strategy. This is reinforced by our case study in Section 8. This strategy correctly captures sequences that occur in the activity stream, but is unable to distinguish between noisy sequences and semantically correct sequences. For instance, a visual examination of the activity stream AS shows the existence of the *P-S-P* sequence, which occurs in the PAG between states $S_6$ and $S_1$. However, in the PAG, this sequence also yields the spurious process path *P-S*.

**3. The B-F($k$, $c$) Strategy.** The model produced by this strategy is shown in Figure 7b.

1. The complexity of the model is also the lowest among all three strategies, comprised of only 7 states. The PAG has captured all correct process paths: *P-Q-R*, and *P-S-T*, as well as sequence *Q-P-R*.

2. It has avoided the *P-S* path that the B-F($k$) PAG produced, and the confidence factor parameter has ensured that the sequence *P-S* is always followed by the activity *T*. It still gives the redundant process path *P-R*.

3. This PAG has fewer redundant paths here; for example, the B-F($k$) PAG, in Figure 6 includes the *P-Q-P-R* sequence twice (linking states 8-9-10-3-2 and 1-2-10-3-2), while a single path would have sufficed. In the B-F($k$, $c$) PAG, such redundancies are avoided, and the *Q-P-R* loop occurs only once (2-10-3-2).

In summary, we observe that the B-F($k$, $c$) PAG has the minimal number of states required to represent the various process paths, and therefore avoids redundancy. It also avoids loops and paths that do not occur frequently enough in the activity streams, and therefore are not a significant part of the process. This appears to validate our intuition that noise elimination is an important goal in modeling BPs. In Section 8 we provide a case study of the discovery of a much more complex process. The conclusions reached from that study reinforce the discussion above, demonstrating the scalability of our approach.

# 8. A Case Study to Validate Our Strategies

In this section, we describe our case-study analysis of the performance of the three strategies upon a complex BP "Provide Computer Support" (PCS), characterizing one of the main goals of the ITSU organization described in Section 2. More specifically, our case-study was conducted as follows:

- We first extracted a process model of this BP by conventional methods, from the agents involved in its enactment as well as other members of the organization responsible for the process design.
- Then we extracted PAGs for the BP using each of the three strategies, and analyzed the PAGS to understand how well the strategies perform at reproducing the process model of step 1, thereby studying their relative process-discovery strengths.

## 8.1. A PAG for Process PCS Using Conventional Methods

The example BP "Provide Computer Support" (PCS) is a complex process of the ITSU, representing the single most important objective of the ITSU services, i.e., serving as the Computing Help Desk for the department. PCS involves several agents and spans a large activity alphabet. An instance of PCS typically takes a few hours to complete, but in some cases, may extend up to one or two working days.

To extract a model using conventional methods, we conducted interviews with agents of the ITSU. The main agents were: the Lab Monitor(s), several Technicians, and the Lab Manager. Interviews with some of the PCS agents yielded the following outline of the process. At a high level, PCS proceeds as follows:

- The process is triggered by a user Support-Request (SR) conveyed by any ITSU user to the Lab Monitor at that time, through phone, e-mail, or in person.
- The Monitor classifies the SR based on which agent it should be sent to, and checks the SR for completeness of information. He/she then forwards it to either the Manager or a Technician (if complete), or back to the user for corrections (if complete). In rare cases, the Monitor may address the SR.
- The Manager chooses the SR to address, and either performs Service Tasks (ST), or sends the SR to a Technician. After completing an SR, the Manager communicates with the user about the SR status, directly or through the Monitor.
- The Technician performs STs, and on completion of the SR, communicates the status to the user, usually through the Monitor.

In order to extract the process activity alphabet, we listed all the agent types involved in this process, such as "Monitor", "Manager", and "Technician". Then, we collected individual activity lists from each of the agents, pertaining to the PCS process only. By collapsing identical activities under a common activity name, we identified the final list of activities and agents, with ample guidance from experienced PCS agents.

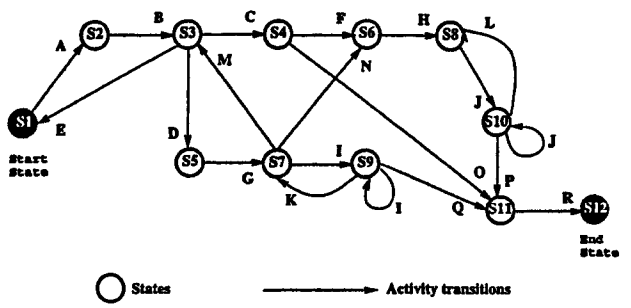
Then each unique activity name-agent pair was given a unique activity label. For example, the activity-type "Forward-SR-To-Technician" is performed both by the Monitor and the Manager. In the former case, it was labeled as "*E*," and in the latter case, as "*N*". Such labeling provided the complete activity alphabet for process PCS, as shown in Table 2. Each activity in the alphabet is characterized by an activity name, agent, and activity label.

With the collective guidance of the PCS agent-team, we produced a conventional process model of the PCS process, in the form of an PAG, as shown in Figure 8. The intuition behind the PAG is as follows: The activity *A* ("Get-SR") signals an incoming user request for Computer Support, and hence, triggers off the process. Therefore state 1, which has an output transition upon *A*, is the *start state* of the process. The agents complete the process by communicating to the user that his/her request has been fulfilled, i.e., by performing activity *R*. Therefore state 12, which has an input transition *R*, is the *end state* of the process. Having identified the start and end states, we can trace several alternative activity sequences, or *paths*, linking states 1 and 12, that all represent possible instances of process PCS. From Figure 8, we can identify the following process paths: (1) *A-B-C-F-H-J-P-R* (2) *A-B-D-G-I-Q-R* (3) *A-B-C-O-R* (4) *A-B-D-G-N-H-J-P-R* (5) *A-B-D-G-M-E-A-B-*. . . (6) *A-B-D-G-I-K-I-I-Q-R*, and (7) *A-B-C-F-H-J-L-J-J-P-R*. Activity loops such as *D-G-M*, *J-L-J*, *I-K-I* and so on, characterize iterations and repetitive sequences that actually occur in PCS. For instance, the PAG shows that, upon *M* (incomplete SR), the process returns to state 3; the state where SRs have been classified, but not yet checked for information accuracy. This PAG provides a very useful benchmark for testing the results of our strategies. Note however, this process took 2 weeks to complete and required extensive time commitments on the part of virtually every agent involved in the process, as well as the analyst team. However, the model yielded by this process appears to be a good one, primarily because the BP was well understood by several agents, and they all agreed that the PAG in Figure 8 represents an accurate model of the PCS process. This was nice, as this gives us a basis to compare the results of the three strategies. In the following two sections, we describe and compare the PAGs generated by the three strategies. These were generated automatically, by coding the algorithms in C and feeding the input activity stream to the programs. Though the running times of these programs are not a major concern, we would like to report that they took, on average, less than 30 minutes. Also note that we did not

**Table 2 Case-Study: Activity Alphabet Description**

| Activity Name | Agent | Label |
|---|---|---|
| Get-SR | Monitor | *A* |
| Classify-SR | Monitor | *B* |
| SR-OK | Monitor | *C* |
| Forward-SR-To-Manager | Monitor | *D* |
| SR-Not-OK | Monitor | *E* |
| Forward-SR-To-Technician | Monitor | *F* |
| Choose-SR | Manager | *G* |
| Choose-SR | Technician | *H* |
| Perform-ST | Manager | *I* |
| Perform-ST | Technician | *J* |
| ST-Not-OK | Manager | *K* |
| ST-Not-OK | Technician | *L* |
| SR-Not-OK | Manager | *M* |
| Forward-SR-To-Technician | Manager | *N* |
| Perform-ST | Monitor | *O* |
| ST-Done-OK | Technician | *P* |
| ST-Done-OK | Manager | *Q* |
| Inform-Customer-SR-Done | All agents | *R* |

**Figure 8 Case-Study: Activity Streams**



make any special attempts to "optimize" the programs—even then, the time savings are enormous.

**8.2. Models of the PCS Process from our Strategies**

In order to generate PAGs for process PCS using our three strategies, we recorded several *activity streams* using established methodologies described in Section 4. Each activity stream was a trace of the repeated enactment of the process over 2 days. We thus captured the following four successive activity streams, during our data-capture period, as shown in Figure 9: Activity stream $AS_1$ served as input to the Probabilistic strategy, and the B-F($k$) strategy. The B-F($k$, $c$) strategy used all the four activity streams. We now describe the PAGs generated by the probabilistic and algorithmic strategies. We first describe the PAGs output by each strategy in Section 8.3, and then compare and contrast the strategies in Section 8.4.

**8.3. Description of the PAGs of the Three Strategies**

**1. PAG: Probabilistic Strategy.**

1. This PAG was generated for a *third-order* activity-probability, i.e., we captured the probabilities of occurrences of all futures, given *three* previous activities. We chose $n = 3$ for the following reason: From the nature of the process as well as the length of the activity stream, it was obvious that sets of three successive activities recurred frequently. Therefore, by following the paths formed by each of these sets, terminated by their most "probable" future, we could extract the most likely process paths.

2. The activity-probability matrix for this strategy, based on $AS_1$ and an $n$ value of 3, is shown in Table 3.

3. We decided to set a threshold probability value of 0.5, for the following reason: in $AS_1$, the maximum frequency of any three-activity sequence is 4; any future of probability less than 0.5 is likely to be insignificant, but futures of equal probability need to be captured.

Figure 10 shows the PAG generated by the probabilistic strategy. There are 21 states in all. In order to ensure that we do not eliminate any of the *legal sequences*, we need to retain some activity edges in the PAG that give rise to some illegal sequences. For example, if edge "*A*" (between nodes $S_2$ and $S_5$) or "*B*"

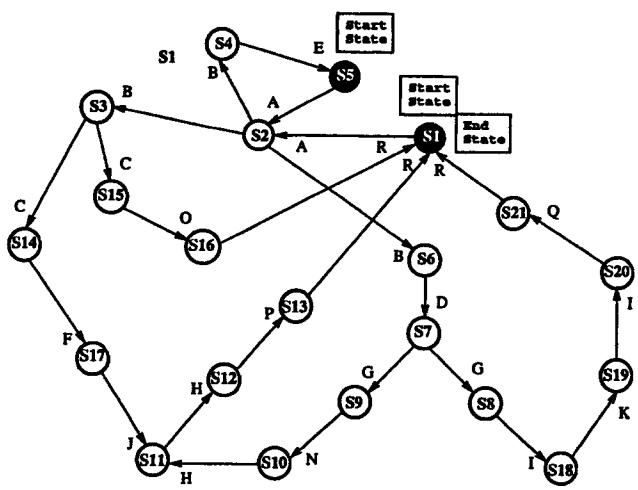
**Figure 9 Case-Study: PAG for Probabilistic Strategy**

| |
|---|
| $AS_1$: *RABEABDGNHJPRABCORABCFHJPRABDGIKIQRABC* |
| $AS_2$: *RABEABDGNHJPRABEABDGMEABDGNHJPRABCFHJP* |
| $AS_3$: *ABCFABDGIQRHJJPRABDGABDGIQRABCFHJPRABC* |
| $AS_4$: *ABCFABDGIQRHJLJPRABCFABDGMEHJPRABDGIIQ* |

**Table 3 Third-Order Activity Probabilities**

| SEQ. | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *RABO* | 0 | 0.6 | 0.2 | 0.2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| *ABE* | 1.0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| *BEA* | 0 | 1.0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| *EAB* | 0 | 0 | 0 | 1.0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| *ABD* | 0 | 0 | 0 | 0 | 0 | 0 | 1.0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| *BDG* | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.5 | 0 | 0 | 0 | 0 | 0.5 | 0 | 0 | 0 | 0 |
| *DGN* | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1.0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| *GNH* | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1.0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| *NHJ* | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1.0 | 0 | 0 |
| *HJP* | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1.0 |
| *JPR* | 1.0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| *PRA* | 0 | 1.0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| *ABC* | 0 | 0 | 0 | 0 | 0 | 0.5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.5 | 0 | 0 | 0 |
| *BCO* | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1.0 |
| *COR* | 1.0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| *ORA* | 0 | 1.0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| *BCF* | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1.0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| *CFH* | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1.0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| *DGI* | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1.0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| *GIK* | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1.0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| *IKI* | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1.0 | 0 |
| *KIQ* | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1.0 |
| *IQR* | 1.0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| *QRA* | 0 | 1.0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Figure 10 Case-Study: PAG for B-F($k$) Strategy**



(between $S_2$ and $S_4$) is eliminated, we can remove the illegal sequence *E-A-B-E*, but the PAG will also lose the legal sequence *B-E-A-B*. But, significantly, the PAG has captured all the legal sequences and all the repeated sequences in $AS_1$, such as *A-B-C-F*, *A-B-D-G*, *H-J-P-R*, and so on.

**2. PAG: B-F($k$) Strategy.**

1. For the B-F($k$) strategy, we used the same activity stream $AS_1$ as we did for the probabilistic strategy (Figure 9). We used a $k$ value of 4; equivalence classes were generated for futures of 4-activity length. The final PAG generated by this strategy, shown in Figure 11, contains 19 states.

2. For the same activity stream, we see that this strategy has captured the same number of process paths (5) with a lesser number of states than the previous strategy.

3. Also, the common sequence of two paths occurs only once here, whereas it was repeated in the probabilistic PAG. For example, path *A-B-C-F-* and *A-B-D-G-N-* both have a single transition upon *H*, from state $S_9$ to $S_{10}$. Therefore there is less redundancy here.

**3. PAG: B-F($k$, $c$) Strategy.** For the B-F($k$, $c$) strategy, we employed all four activity streams $AS_1, \ldots, AS_4$, given in Figure 9. This was in order to select histories for the Final Equivalence classes, based on their *confidence factor*. The final PAG is shown in Figure 12.

1. In this case, we chose a $k$ value of 4, as in the B-F($k$) strategy, and a confidence factor $c$ of 0.5. The $c$ value chosen here tries to ensure that, for any given future, only those histories are selected that precede this future at least half the number of times they occur. In the activity streams used, we see two such pairs of histories that are repeated twice, giving a confidence factor of 0.5 each. Therefore, these histories were both included in the FECs, and are reflected in the PAG. The histories are: *ABCFABDGIQRHJ*, and *RABEABDGNHJPRAB*. (A lower confidence factor would have caused both the histories to be omitted, and the PAG would not have captured some important process paths such as *A-B-C-O-R*).

2. Since B-F($k$, $c$) uses several activity streams, it captures process paths and iteration loops that are missed by the other two strategies, but the PAG still has just 21 states (including a distinct start state): equal to that of the probabilistic strategy, and just two more than B-F($k$).

### 8.4. Comparison of the Strategy Results

We present a comparison of the three strategies with respect to how closely they approximate the conventional PCS model (see Figure 8).

**1. The Probabilistic PAG (see Figure 10).**

1. The PAG has 21 states, with two start states, and one end state.

2. It extracts all of the possible process paths derived from the conventional PAG (see Figure 8). The corresponding process paths were described in Section 8.1,

**Figure 11 Case-Study: PAG for B-F($k$) Strategy**

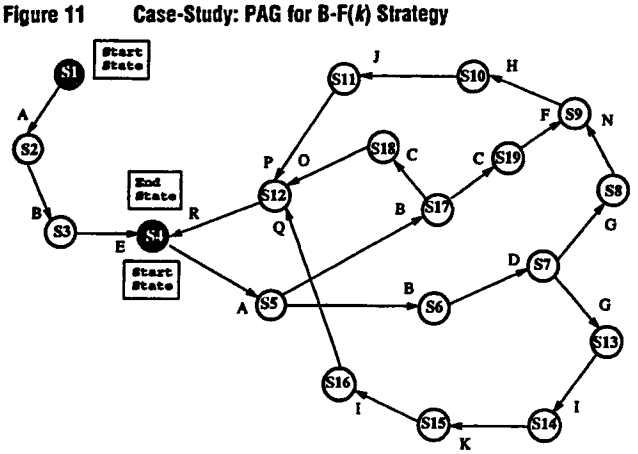
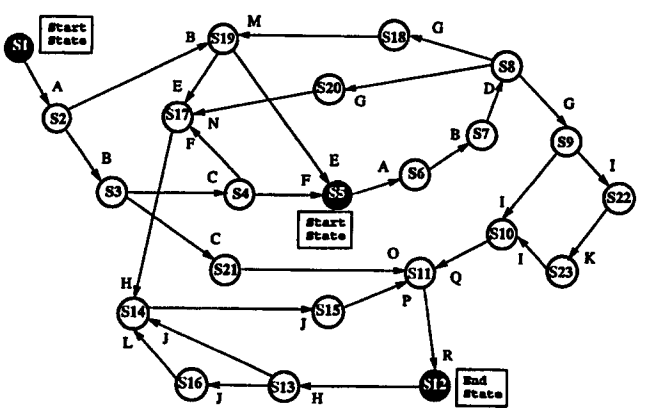



**Figure 12 Case-Study: PAG for B-F($k$, $c$) Strategy**



excepting the path *A-B-D-G-I-Q-R*, and the somewhat infrequent loops *H-J-J*, *G-I-I*, *D-G-M*, and *J-L-J*. But it can be observed that these loops are not present in $AS_1$, and the problem of missing sequences arises since we can use only a single activity stream as input.

3. There are some redundant transitions and states, causing replicated sequences; for instance, edge "*A*" and hence, sequence *A-B* occurs four times, though it is a common history for four paths in the PAG. This is because, for each legal sequence in the probability matrix, a new set of edges is created, regardless of whether part of the sequence already exists in the graph.

4. Unlike the algorithmic strategies, there is no merge-state step here to reduce the above redundancy, and hence, the resulting PAG is less compact than the B-F(*k*) PAG.

**2. B-F(*k*) PAG (see Figure 11).**

1. This PAG has a fewer number of states (19), with respect to the probabilistic strategy.

2. The PAG has captured the same paths as the probabilistic PAG, and has also missed the *A-B-D-G-I-Q-R* path as well as the smaller loops.

3. The edge "*A*" and sequence *A-B* occur only twice, showing a reduction of repetitions with respect to the probabilistic strategy. This is a result of the merge-state technique introduced into the strategy.

4. The algorithm produces a more compact and concise PAG than the probabilistic strategy. But since, here too, we can use only a single activity stream, the chances of missing some legal sequences and path extensions are higher than in the case of the B-F(*k*, *c*) strategy.

**3. B-F(*k*, *c*) PAG (see Figure 12).**

1. As a result of using four input activity streams in Figure 9, the B-F(*k*, *c*) PAG has captured more process paths than either of the above, though it has just two states more than the probabilistic PAG.

2. In addition to the paths mentioned above, this PAG also captures the path *A-B-D-G-I-Q-R*. The smaller loops or extensions missed by the other two strategies occur in this PAG; these are: *G-M-E-A*, *D-G-I-Q-R*, *H-J-J-P-R*, and *H-J-L-J-P-R*.

3. The merge-state technique ensures that a minimal number of states is required to capture all the possible paths. The confidence factor of 0.5 also ensures that, when the activity streams include sequences with equally probable futures, both of these paths are captured. This can be seen, in the case of sequence *RABEABDGNHJPRAB*, in $AS_1$ and $AS_2$.

4. Thus, in terms of the process paths captured, the comparison clearly illustrates that the B-F(*k*, *c*) strategy offers the closest and most complete PAG of a given process.

## 9. Why Did We Choose Probabilistic Strategies?

The reader may have noticed that the three strategies outlined previously are all probabilistic. A relevant question therefore is why we chose this strategy rather than, say, a machine learning based strategy. We provide a three part answer to this question:

• It is perfectly legitimate to use machine learning based strategies such as neural networks or genetic algorithms (see Michalski et al. (1983) for a nice introduction to machine learning) for this problem.

In fact, in many areas such as *information retrieval*, there are two clear cut philosophies, a probabilistic school of thought and a machine learning school of thought. Consider, for instance, the well-known problem of *classification*, in much vogue currently owing to its applicability to the field of *data mining*. There exists a *huge* literature in classification from both the probabilistic as well as the machine learning perspectives (see, e.g., Wong and Yao (1993) for a probabilistic approach to this problem and Wu et al. (1995) for a machine learning approach). This is true in several other research areas where the fundamental problem is the "discovery" of something based on certain inputs. Our problem definitely fits this criteria. However, to the best of our knowledge, there is *no* consensus on one approach being "superior" to the other. If there existed machine learning based methods of BP discovery, it would have been incumbent upon us to make a comparison. However, even after a careful search, we were unable to identify any such work. In fact, we were unable to find *any* work on this area, machine learning or otherwise. However, exploring machine learning to achieve BP discovery is definitely a worthwhile research area and we would be quite eager to see how

such strategies would perform in comparison to us. We doubt, however, that unilateral superiority could be claimed by either approach.

• In our opinion, one of the interesting intuitions that drove this work was the commonality between the grammar discovery problem and the process discovery problem. The work on grammar discovery, in large part was probabilistic.

• Finally, the authors of this paper are not as familiar with machine learning as with stochastic process theory—thus, probabilistic methods appeared more natural to apply.

## 10. Conclusion

The modeling of AS-IS processes is well recognized as a necessary first step of business process reengineering (BPR). Unfortunately no tools exist that would enable organizations to undertake such an endeavor themselves. Such tools are impossible to create, as no systematic procedures exist to perform AS-IS model discovery. As a result, corporations wishing to reexamine their business processes have to rely on external expertise to perform this activity. These experts typically follow ad-hoc or "home grown" methods, usually consisting of a large number of meetings with individuals and groups. As a result, such efforts consume large amounts of time and money.

Given this context, the primary contribution of this paper is that it postulates a number of *systematic* procedures to extract AS-IS process models. We have developed AS-IS process discovery tools by programmatically implementing these procedures. We have demonstrated that these procedures work reasonably well, through two "real-life" discovery applications reported earlier in the paper. While the primary contribution is extremely practical, we also make a significant theoretical contribution, by recognizing the commonality between the grammar discovery problem and the BP discovery problem. As a result of this mapping we are able to apply variants of established FSM synthesis algorithms for our purposes. We also recognize certain differences between the classical grammar discovery scenario and our problem, e.g., the presence of noise. These recognitions aid us in smartly modifying established algorithms to suit our case. Finally, this paper appears to be the first one to look at automating BP discovery—this itself, we believe, is a contribution of this work. Our methods, if adopted, promise to save organizations substantial amounts of money and time.

For future work we are pursuing several avenues of refining our algorithms to discover control flows. In particular we have initiated a project to discover contingency relationships between tasks and conditional branching from a particular state. Our intuition is that this should be possible by additional processing of the activity streams, i.e., introducing additional logic to our current algorithms. We have certain preliminary ideas regarding how to proceed. For instance, it appears certain that none of our "single-stream" algorithms are particularly well suited, as a single sequence may not capture contingency actions or conditional branchings. Contingency points (i.e., states leading to contingency actions) may be discovered by augmenting the event definition that we currently use (which conforms to event definitions available in the open literature), with additional semantics. For example, instead of simply classifying events as begin or end we can attempt to subclassify the end events as *end-with-success* (EWS) events and *end-with-failure* (EWF) events. Now, we could examine the input streams paying special attention to histories which have prefixes ending with EWF events. Subsequently, one way to discover contingency actions would be to identify "different" activity sequences (i.e., histories) following the same EWF terminated prefix. Similar avenues could be pursued for conditional branchings.

### References

Abeysingle, G. and K. Phalp, "Combining Process Modeling Methods," *Information and Software Technology*, 39, 2 (1997), 107–124.

Angluin, D. and C. Smith, "Inductive Inference: Theory and Methods," *ACM Computing Surveys*, 15, 3 (1983), 237–269.

Anton, A., M. McCracken, and C. Potts, "Goal Decomposition and Scenario Analysis in Business Process Reengineering," in *CAISE '94*, Springer 1994, 94–104.

Biermann, A. and J. Feldman, "On the Synthesis of Finite State Machines from Samples of Their Behavior," *IEEE Trans. on Computers*, 21, 6 (1972), 592–597.

Blyth, A., J. Chudge, J. E. Dobson, and M. R. Strens, "Ordit: A New Methodology to Assist in the Process of Eliciting and Modelling Organizational Requirements," in *Conf. on Organizational Computing Systems*, Supporting Software Development Organizations, ACM 1993, 216–227.

Bradac, M., D. Perry, and L. Votta, "Prototyping a Process Monitoring Experiment," *IEEE Trans. on Software Engineering*, 20, 10 (1994), 774–784.

Cook, J. and A. Wolf, "Automating Process Discovery Through Event-Data Analysis," in *Proc. 17th Intl. Conf. on Software Engineering (ICSE17)*, April 1995, 73–82.

Davenport, T., *Process Innovation—Reengineering Work through Information Technology*, Harvard Business School, Boston, MA, 1993.

Dinkhoff, G., V. Gruhn, A. Saalmann, and M. Zielonka, "Business Process Modeling in the Workflow Management Environment Leu," in *Proc. 13th Intl. Conf. on the Entity-Relationship Approach*, Springer, Manchester, U.K. December 1994a, 46–63.

——, ——, ——, ——, and P. Loucopoulos, "Business Process Modeling in the Workflow Management Environment Leu," in *Proc. 13th Intl. Conf. on the Entity-Relationship Approach*, Springer-Verlag, Manchester, U.K. December 1994b, Implementation.

Gill, A., "Realization of Input-Output Relations by Sequential Machines," *J. Assoc. Computing Machinery*, 13, 1 (1962), 33–42.

Ginsburg, S., *An Introduction to Mathematical Machine Theory*, Reading, MA, 1962.

——, *The Mathematical Theory of Context-Free Languages*, McGraw-Hill, New York, 1966.

Hammer, M. and J. Champy, *Reengineering the Corporation: A Manifesto for Business Revolution*, Harper Business, New York, 1993.

—— and S. Stanton, *The Reengineering Revolution—A Handbook*, Harper Business, New York, 1995.

IBM Corporation, "Flowmark—A Workflow Modeling and Process Monitoring Software," World Wide Web Site, 1996, http://www.software.hosting.ibm.com/workgroup/flowmark/exmn0mst.htm.

Kamath, M. and K. Ramamritham, "Bridging the Gap between Transaction Management and Workflow Management," NSF Workshop on Workflow and Process Automation in Information Systems, 1996. http://lsdis.cs.uga.edu/NSF-workflow/kamath.html.

Kellner, M., P. Feiler, A. Finkelstein, T. Katayama, L. Osterweil, M. Penedo, and H. Rombach, "Software Process Modeling Example Problem," in *Proc. 6th. Intl. Software Process Workshop*, October 1990, 19–29.

Kueng, P., P. Bichler, and M. Schrefl, "Business Process Modeling: A Goal Based Approach," *Information Management*, 11, 2 (1996), 40–50.

Kuo, D., M. Lawley, L. Chengfei, and M. Orlowska, "A Model for Transactional Workflows," *Australian Computer Science Comm.*, 18, 2 (1996), 139–146.

LeBlanc, R. and A. Robbins, "Event-Driven Monitoring of Distributed Programs," in *Proc. 5th Intl. Conf. on Distributed Computing Systems*, IEEE Computer Society, May 1985, Denver, CO., 515–522.

Manley, J. H., "Enterprise Information Systems Modeling for Continuous Improvement," *Computers and Industrial Engineering*, 31, 1–2 (1996), 273–276.

Martinsons, M., "Radical Process Innovation Using Information Technology: The Theory, the Practice and the Future of Reengineering," *Intl. J. Information Management*, 15, 4 (1995), 253–269. Review.

Masaaki, K., *Markov Processes for Stochastic Modeling*, Chapman and Hall, New York, 1997.

Medina-Mora, R., T. Winograd, R. Flores, and F. Flores, "The Actionworkflow Approach to Workflow Management Technology," in *Proc. Conf. on Computer-Supported Cooperative Work*, ACM, Toronto, 1992, 281–288. Implementation.

Michalski, R. S., J. G. Carbonell, and T. M. Mitchell, *Machine Learning: An Artificial Intelligence Approach*, Tioga, Palo Alto, CA., 1983.

Miclet, L. and J. Quinqueton, in G. Ferrate, T. Pavlidis, A. Sanfeliu, and H. Bunke, Eds., "Learning from Examples in Sequences and Grammatical Inference," vol. 45 of NATO ASI series F: Computers and System Sciences, Springer-Verlag, New York, 1988, 153–171.

Nerode, A., "Linear Automation Transformations," in *Proc. American Math. Soc.*, 9 (1958), 541–544.

Ngu, A., T. Duong, and U. Srinivasan, "Modeling Workflow Using Tasks and Transactions," NSF Workshop on Workflow and Process Automation in Information Systems, 1996. http://lsdis.cs.uga.edu/activities/NSF-workflow/Final/ngu/work3.html.

Rajapakse, J. and M. Orlowska, "On Specification of Transactional Workflows for Integrated Systems," in *Proc. InterSymp-95 of the International Conf. on Systems Research*, 1995.

Rusinkiewicz, M. and A. Sheth, "Specification and Execution of Transactional Workflows," in *Modern Database Systems: The Object Model, Interoperability, and Beyond*, ACM Press, New York, 1995, 592–620. Implementation.

Sato, R. and H. Praehofer, "A Discrete Event System Model of Business Systems—A Systems Theoretic Foundation for Information Systems Analysis," Part 1, *IEEE Trans. on Systems, Man and Cybernetics*, 27 Part A(1), January (1997).

Weissenfels, J., D. Wodtke, G. Weikum, and A. Kotz-Dittrich, "The Mentor Architecture for Enterprise-Wide Workflow Management," NSF Workshop on Workflow and Process Automation in Information Systems, 1996. http://lsdis.cs.uga.edu/activities/NSF-workflow/Final/mentor.html.

Wolf, A. and D. Rosenblum, "A Study in Software Data Capture and Analysis," in *Proc. Second Intl. Conf. on the Software Process*, Berlin, Germany, February 1993, IEEE Comput. Soc. Press, 115–124.

Wong, S. K. M. and Y. Y. Yao, "A Probabilistic Method for Computing Term-by-Term Relationships," *J. American Society for Information Science*, 44, 8 (1993), 431–439.

Wu, C., M. Berry, S. Shivakumar, and J. McLarty, "Neural Networks for Full Scale Protein Sequence Classification: Sequence Encoding with Singular Value Decomposition," *Machine Learning*, 21, 1–2 (1995), 177–193.

Yu, E., "Models for Supporting the Redesign of Organizational Work," in *Proc. Conf. on Organizational Computing Systems*, August 1995, 225–236. http://www.cs.utoronto.ca/eric/publications.html.

——, P. Du Bois, E. Dubois, and J. Mylopoulos, "From Organization Models to System Requirements—A "Cooperating Agents" Approach," in *Proc. 3rd Intl. Conf. on Cooperative Information Systems*, Vienna, Austria, May 1995, 194–204.

—— and J. Mylopoulos, "Towards Strategic Actor Relationships for Information Systems Development—With Examples from Business Process Reengineering," in *Proc. 4th Workshop on Information Technologies and Systems*, Vancouver, Canada, December 1994, 21–28. http://www.cs.utoronto.ca/eric/publications.html.

*Amitava Dutta, Associate Editor. This paper was received on February* 21, 1997, *and has been with the author* 14 *months for* 2 *revisions.*