

A Machine Learning Approach to Workflow Management

Joachim Herbst

DaimlerChrysler AG, Research and Technology
P.O. Box 2360, 89013 Ulm, Germany
joachim.j.herbst@daimlerchrysler.com

Abstract. There has recently been some interest in applying machine learning techniques to support the acquisition and adaptation of workflow models. The different learning algorithms, that have been proposed, share some restrictions, which may prevent them from being used in practice. Approaches applying techniques from grammatical inference are restricted to sequential workflows. Other algorithms allowing concurrency require unique activity nodes. This contribution shows how the basic principle of our previous approach to sequential workflow induction can be generalized, so that it is able to deal with concurrency. It does not require unique activity nodes. The presented approach uses a log-likelihood guided search in the space of workflow models, that starts with a most general workflow model containing unique activity nodes. Two split operators are available for specialization.

1 Introduction

The success of today's enterprises depends on the efficiency and quality of their business processes. Software based tools are increasingly used to model, analyze, simulate, enact and manage business processes. These tools require formal models of the business processes under consideration, which are called workflow models in the following. Acquiring workflow models and adapting them to changing requirements is a time consuming and error prone task, because process knowledge is usually distributed among many different people and because workflow modeling is a difficult task, that needs to be done by modeling experts (see [1,5] or [9]). Thus there has been interest in applying machine learning techniques to induce workflow models from traces of manually enacted workflow instances. The learning algorithms, we are aware of, share some restrictions, that may prevent them from being used in practice. They either apply grammatical inference techniques and are restricted to sequential workflows [5,9] or they allow concurrency but require unique activity nodes [1,6].

2 Definitions

In the following we define the terms *workflow model* and *workflow instance*. This is essential for a description of the induction task. A workflow model is a

formal explicit representation of a business process, describing how this process is (or should be) performed. It decomposes the process into elementary activities and defines their control and data flow. The activities $A = \{a_1, \dots, a_n\}$ of the process are specified in terms of their required resources and actors. Different formalisms have been proposed for workflow modeling. Within this paper we are using the ADONIS modeling language [3]. According to the ADONIS modeling language a workflow model can be defined as follows: A *workflow model* is a tuple $M = (V_M, f_M, R_M, g_M, P_M)$, where $V_M = \{v_1, \dots, v_{n_M}\}$ is a set of nodes, $START_M, ACT_M, DEC_M, SPLIT_M, JOIN_M, END_M$ is a partition of V_M , $f_M : ACT_M \rightarrow A$ is the activity assignment function, that assigns an activity to each activity node, $R_M \subseteq (V_M \times V_M)$ is a set of edges, $P_M : R_M \rightarrow [0, 1]$ assigns a transition probability to each edge and $g_M : R_M \rightarrow COND$ assigns a condition to each edge. This definition is incomplete, as it concentrates on the behavioral and functional view (see [7]) on a workflow model. For a complete definition describing also the organizational and informational view [7] as well as a discussion of additional syntactical rules and the semantics of the modeling language we refer to [3]. For our purposes the above definition and figure 1 showing the graphical representation of the node types and a brief explanation of their semantics should be sufficient. An example for a workflow model is given in figure 2.







Graphical Representation	Node Set	Explanation
	START	Starting node of a workflow model.
	ACT	An activity node.
	SPLIT	An m of n split. (m of n successors may be activated)
	JOIN	Join nodes synchronize the concurrent threads of their corresponding split nodes
	DEC	A decision node. (Exactly 1 of n successors may be activated)
	END	End node of a workflow model.

Fig. 1. ADONIS node types

A *workflow instance* is a tuple $e = (K_e, f_e(), \leq_e)$, where $K_e = \{k_1, \dots, k_{n_e}\}$ is a set of nodes, $f_e : K_e \rightarrow A$ is the activity assignment function, which assigns an activity to each node and \leq_e is a partial order on K_e . Workflow instances represent a completed business cases. The nodes describe the activities, which were executed to complete a business case and the partial order describes the temporal order of their execution. For the sake of clarity, we define only those components, of a workflow instance which are relevant for this paper. Two exam-

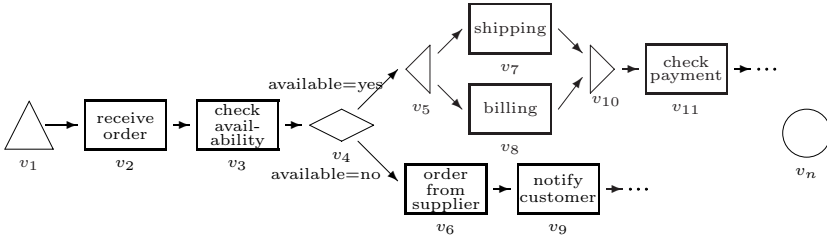


Fig. 2. Part of a simple ADONIS workflow model

ples for workflow instances are shown in figure 4. Activity nodes are represented by boxes, which are labeled with the values of the activity assignment function.

3 Inducing Workflow Models

3.1 Characterization and Decomposition of the Induction Task

The induction task to be solved can be characterized as follows: Given a multiset of workflow instances E , find a good approximation M of the workflow model M_0 , that generated E . Of course M_0 need not exist. It is simply a modeling hypothesis. We have decomposed the induction task into two subtasks:

- Induction of structure - within this subtask the nodes, the edges, the activity assignment function and the transition probabilities of M are induced.
- Induction of conditions - where possible, local conditions for transitions following a split or a decision node are induced.

In this contribution we focus only on the induction of the structure. The induction of conditions can be done using standard decision rule induction algorithms such as C4.5 [12] as explained in more detail in [9].

3.2 Problem Classes

To allow a comparison between different workflow induction algorithms reported in the literature, we have defined four problem classes. These are defined in terms of two characteristics of the unknown workflow model M_0 . The first characteristic is sequentiality. A workflow model is strictly sequential, if it does not contain any split or join nodes. The second characteristic is a characteristic of the activity assignment function f_{M_0} . As we will see, it is a difference whether f_{M_0} is injective or not. If the activity assignment function is injective, then the unknown workflow model M_0 contains unique nodes for each observed activity. Using these two characteristics the four problem classes shown in figure 3 can be defined.

Actually it would be sufficient, to solve the induction task for the most general problem class, which contains all other problem classes. But we are not aware of

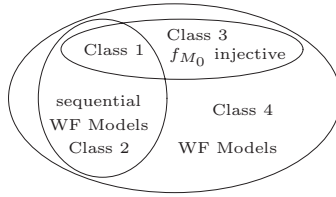


Fig. 3. Problem classes

any induction algorithm in the literature, that attempts to solve the induction task for this class. In this paper we will shortly discuss the induction of sequential workflow models and in more detail we will explain how this algorithm can be generalized to provide a solution to the problem classes three and four.

3.3 Sequential Workflow Models: Problem Classes 1 and 2

The structure of sequential workflow models can be represented by stochastic finite state automata (SFA) and sequential workflow instances can be seen as strings over a finite alphabet. Each symbol in this alphabet corresponds to an activity of the workflow instance. Thus the problem of sequential workflow structure induction can be reduced to the problem of inducing SFAs from a positive sample of strings. This problem has already been addressed in the grammatical inference community (see e.g. [11]) and some algorithms like e.g. ALERGIA [4] or Bayesian Model Merging [14], have been proposed. In [9] we present two algorithms for sequential workflow induction. The first one follows a specific to general approach. It is a variation of the Bayesian Model Merging [14], using the log-likelihood of the workflow model as a heuristic. The second one follows a general to specific approach. For specialization it applies a split operator that splits one node into two nodes assigned to the same activity. Search starts with a most general model, containing unique nodes for each observed activity. The solution to problem class 4, which we present below, follows the same basic principle.

3.4 Concurrent Workflow Models with Unique Activity Nodes: Problem Class 3

Let's now turn to concurrent workflow models having unique activity nodes. For each activity $a_i \in \{a_1, a_2, \dots, a_n\}$ we observe, we create a unique node v_i with $f_M(v_i) = a_i$. This gives us the set of activity nodes $\text{ACT}_M = \{v_1, \dots, v_n\}$ of the workflow model M . Whenever we observe the occurrence of an activity a_i , we can identify the corresponding activity node v_i of M . This allows us to talk about the "occurrence of a node v_i within an instance e ".

But as the workflow model may contain concurrent threads we may not determine the activity node, whose completion triggered the current observed activity, as easily as in the sequential case, where we considered the immediate predecessor to be the cause for an observed activity (see [9]). This is not adequate

in case concurrent threads of control are possible. First of all the cause for an observed activity is not necessarily its immediate predecessor and also there may be more than one cause for an activity (e.g. after a join construct). This is shown in figure 4. The workflow instances e_1 and e_2 may have been generated by the workflow model M_0 . In this case the cause for activity D within instance e_2 is activity C, and not its immediate predecessor B.

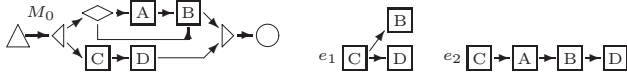


Fig. 4. Unknown model M_0 and observed workflow instances e_1 and e_2

Before we can add edges to M , we must find the cause for an observed activity. This leads us directly to the task of detecting dependencies between activities. This is done by analyzing the temporal relationships between activities. For the following definition to be well defined, we need to assume, that the unknown model M_0 is acyclic. We will later eliminate this restriction. This assumption assures that no activity occurs more than once within a workflow instance. We can now define the dependency graph as the directed graph $G_{\text{dep}} = (V_M, R_{\text{dep}})$ with

- $V_M = \text{START}_M \cup \text{ACT}_M \cup \text{END}_M$ with $\text{START}_M = \{v_0\}$, $\text{END}_M = \{v_{n+1}\}$
- $R_{\text{dep}} = \{ (v_i, v_j) \mid \forall e \in E (v_i, v_j \text{ appear in } e) \Rightarrow v_i \text{ precedes } v_j \text{ in } e \}$, v_0 and v_{n+1} implicitly occur within every e . v_0 is a predecessor of any node within every e and every node occurring within e is a predecessor of v_{n+1} .

The dependency graph can be determined in one pass over the sample E . If we observed all possible instances that could be generated from M_0 the dependency graph shown in figure 5 would be found. We also define dependency graphs $G_e = (V_M(e), R_{\text{dep}}(e))$ for each instance e . G_e is the subgraph of G_{dep} containing only those nodes occurring in e and all edges between them. The dependency graphs for the instances e_1 and e_2 are depicted on the top right of figure 5. Using the dependency graphs G_e , we now determine the cause graphs $(G_e)_*$. The cause graph $(G_e)_* = (V_M, R_{\text{dep}}(e)_*)$ is the transitive reduction of G_e . The transitive reduction of a directed graph G is defined as a minimal subgraph of G having the same transitive closure as G . In this case the transitive reduction of G_e is unique because G_e is acyclic and it can be efficiently determined, because a topological ordering of the nodes in G_e is indirectly given by the partial order \leq_e of the workflow instance. The cause graphs for each workflow instance can be calculated in a second pass over the sample E . The cause graphs for e_1 and e_2 are shown on the bottom left of figure 5. We can now determine the set of edges R_M of M as $R_M = \bigcup_{e \in E} R_{\text{dep}}(e)_*$.

Let's drop the assumption that M_0 is acyclic. Now an activity a_i may appear more than once within an instance e . We simply distinguish different occurrences

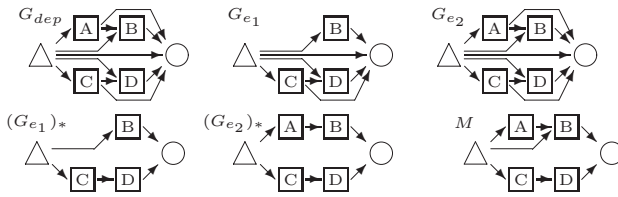


Fig. 5. Dependency graphs, cause graphs and induced model

of one and the same activity within one instance by adding an index (1st, 2nd, 3rd, ... occurrence). We then apply the same algorithm and treat different occurrences of the same activity as different activities until the edges of the model have been determined. At this point all nodes belonging to the same activity are merged to one node, that inherits the edges of all merged nodes.

To complete M we finally add explicit control flow constructs (Decision, Split and Join) to the model M where necessary. This step is not as trivial as in the sequential case (compare [9]), because dependencies between different edges must be analyzed. Within this paper we will not elaborate on this task any further.

3.5 Concurrent Workflow Models in General: Problem Class 4

For problem class 4 M_0 may contain more than one node for a specific activity. The basic idea of our solution is the same as the splitting approach presented in [9] for sequential workflows. One starts with the most general model, generated by the algorithm of the previous section, called `induceUniqueNodeModel()` in the following. This is like assuming that M_0 is in problem class 3. The most general model is specialized using split operators. The selection of the state to split and of other parameters is guided by the log-likelihood per sample. In our prototype we are using beam-search as search algorithm. A larger model (containing more nodes) is preferred over a smaller model, only if the log-likelihood per sample is larger than some user defined threshold LLH_{\min} .

Probability of a Sample The likelihood heuristic requires to estimate the probability of E given M . For this purpose one could describe all outgoing edges of a node v_i as a n of m selection. To distinguish decision nodes allowing only a 1 of m selection from split nodes we decided to use better estimation, that considers clusters of nodes. These are defined in a way that all nodes v_j sharing a common cause v_i within any instance are contained in a common cluster C_{ik} ¹. This idea is shown in figure 6. In the example of figure 6 the probability that the activities B and C follow the activity A would be calculated for example as $0.5 \cdot (1 \cdot 0.5 \cdot (1 - 0.5))$. The transition probabilities are estimated by the empirical counts.

¹ This is still a simplification because dependencies may be more complex. One might for example be interested in finding exclusive successors within one cluster

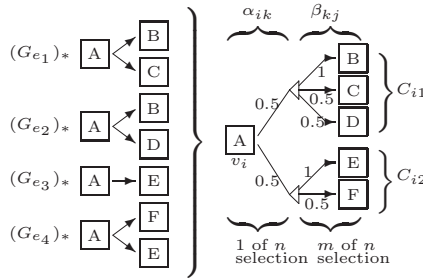


Fig. 6. Clustering of nodes

Specialization of Concurrent Workflow Models For sequential workflow induction [9] we defined the split operator as an operator on the workflow model. The effects of a split operation on a concurrent model are not restricted to the incoming and outgoing edges of the node that is split. Global effects are possible if the split operation changes the dependencies. It is thus not clear how to find a simple description for a split operator on concurrent workflow models. To prevent these difficulties, we define the split operator as an operator on the workflow instances, that introduces an artificial distinction between certain occurrences of the activity that is split. After a split operator has been applied to all instances in E , `induceUniqueNodeModel()` called with the changed instances E to return a specialized model.

Split Operators For the specialization of the workflow model we initially defined one split operator $\text{Split}_{\text{Cause}}(e, a_i, a_j, (G_e)_*)$ as:

$$\forall k \in K_e \text{ with } f_e(k) = a_i \text{ let } f_e(k) := \begin{cases} a'_i & \text{if } a_j \text{ is a cause of } a_i \\ a''_i & \text{otherwise} \end{cases}$$

While one split operator based on the cause of a node is sufficient for sequential workflows, it is sometimes not applicable for concurrent workflows. When multiple activity nodes are allowed, dependencies are often not identified correctly until the right degree of specialization has been reached. This may have the consequence that a certain cause for an activity, can not be correctly identified. If this cause is necessary to correctly distinguish different occurrences of an activity, $\text{Split}_{\text{Cause}}$ fails. To deal with this problem, we define a second split operator $\text{Split}_{\text{History}}(e, a_i, a_j)$ as:

$$\forall k \in K_e \text{ with } f_e(k) = a_i \text{ let } f_e(k) := \begin{cases} a'_i & \text{if } a_j \text{ is a predecessor of } a_i \text{ in } e \\ a''_i & \text{otherwise} \end{cases}$$

4 Related Work

In [1] an approach called process mining, based on the induction of directed graphs, is presented. It is restricted to problem class 3 and very similar to our

approach for this problem class. The main differences to our approach lie in the representation of workflow instances as strictly ordered sets of activities and in the way dependencies are defined and determined. Another approach that is also restricted to problem class 3 is presented in [6]. It uses three different metrics for the number, frequency and regularity of event sequences to estimate a model of the concurrent process. In their previous work [5] the authors applied different grammatical inference algorithms to sequential workflows.

Different approaches combining machine learning and workflow management techniques are presented by Wargitsch [15] and by Berger et. al. [2]. Both are using completed business cases to configure new workflows. While Wargitsch employs a case-based reasoning component for the selection of an appropriate historical case, Berger et. al. are using a neural network approach.

Workflow induction has some similarity with the mining of temporal patterns presented in [13] or [10]. But while we are trying to find one structure in a relatively structured event trace, these approaches are trying to find all frequent structures and they are applicable only for unstructured event traces, as their performance scales exponentially with the size of the largest structure found.

5 Prototype and Experiences

We have realized a research prototype using the business process management system ADONIS both as a front end for the generation of artificial workflow instances and as a back end for the layout generation and visualization of the induced workflow models. We applied this prototype to workflow traces generated by different types of workflow models. Some of these models are from the literature (see e.g. [1] or [6]), some of them are workflow models we have defined and others are real workflow models we have encountered within workflow projects at DaimlerChrysler. In [8] we describe the application of our approach to a simplified release process of the Mercedes Benz passenger car department. Tables 1 and 2 show comparisons with process mining [1] and with process discovery [6]. As the original samples were not available, we generated our own samples. This of course has an influence on the results.

Table 1. Workflow splitting applied to workflow models reported in [1]

Model	Nodes / Edges	Nr. splits	Nr. samples	Process Mining		Workflow Splitting	
				time	correct?	time	correct?
Upload_And_Notify	11/11	0	134	11.5s	yes	0.8s	yes
StressSleep	18/27	0	160	11.7s	yes	5.6s	yes
Pend_Block	10/11	0	121	6.3s	yes	0.8s	yes
Local_Swap	14/13	0	24	5.7s	yes	0.2s	yes
UWI_Pilot	11/11	0	134	11.8s	yes	0.8s	yes

The comparison with process mining shows that exactly the same models are found, which is not surprising as the algorithms are very similar. The improvement concerning the performance might be caused by the slightly different definition of dependency we are using, which allows a more efficient algorithm for dependency detection. Actually our approach should be less efficient, because process mining is restricted to workflow models of problem class 3 and does not try to split any nodes. The models described in [6] were initially not identified correctly by our approach. They contained some incorrect edges. The reason for these incorrect edges is that both models contain concurrent activities within cycles. With some probability only a few samples are available for those workflow instances with the highest number of iterations over a certain cycle. In this case it is likely that not all possible orderings of activities are observed for this highest iteration. This may lead to an incorrect dependency graph and as a consequence to an incorrect model. As these incorrect edges are characterized by a probability close to zero they can be identified and removed from the model. This enables our approach to induce these models correctly as well.

Table 2. Workflow splitting applied to workflow models reported in [6]

Model	Nodes / Edges	Nr. splits	Nr. samples	Process Discovery		Workflow Splitting	
				time	correct?	time	correct?
simple concurrent Process	11/12	0	300	?	yes	126.8s	(yes)
complex concurrent Process	22/26	0	150	?	yes	194.7s	(yes)

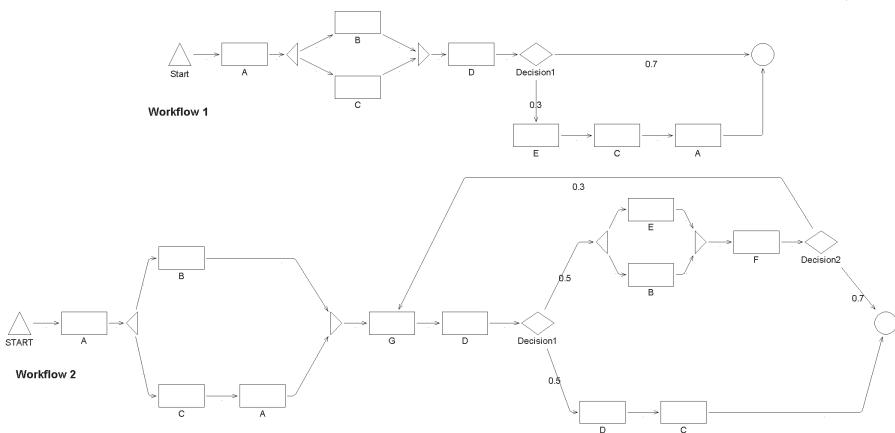


Fig. 7. Two Workflow Models used for evaluation

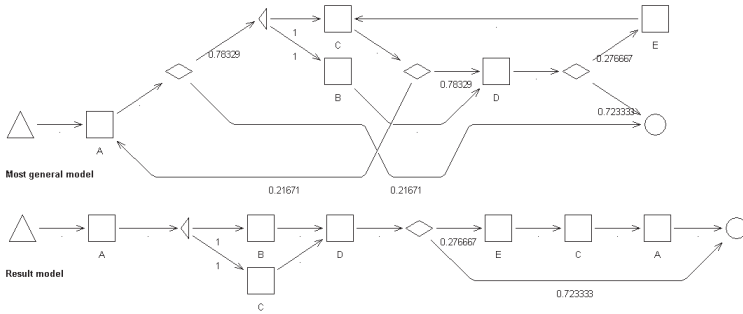


Fig. 8. Workflow 1: Most general model and result model after applying $\text{Split}_{\text{Cause}}(e, A, C, (G_e)_*)$ and $\text{Split}_{\text{Cause}}(e, C, E, (G_e)_*)$

The workflow models presented in [1] and [6] are all located in problem class 3. To evaluate the specialization procedure we also applied our approach to workflow models of problem class 4. Two examples for such workflow models are given in figures 7.

When observing a large enough sample generated from workflow 1 the most general model depicted at the top of figure 8 would be induced. Given the right choice for LLH_{\min} , after one intermediate step our search procedure would return the model shown at the bottom of figure 8. Any further split operations lead only to a small improvement of the log-likelihood per sample.

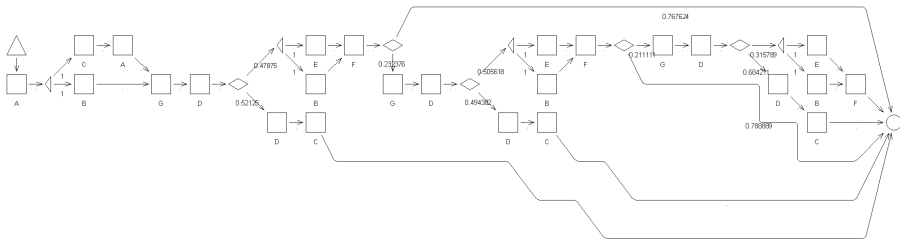


Fig. 9. Workflow 2: Overly specific model using $\text{LLH}_{\min} = 0$

The degree of specialization depends on the user defined threshold LLH_{\min} . Overly specialized models will for example be found if this threshold is too small. The effect of overspecialization is shown in figure 9. The cycle present within workflow 2 of figure 7 has been “unrolled”. Figure 10 shows the log-likelihood per sample for those models on the search path from the most general model to the model of figure 9. As you can see, only the first four (from left to right) split

operations significantly improve the log-likelihood per sample. After the fourth split, the log-likelihood per sample remains nearly constant. Thus the threshold LLH_{\min} must be chosen within the right range close to zero, so that the search stops after the fourth split and returns the correct model shown in figure 11.

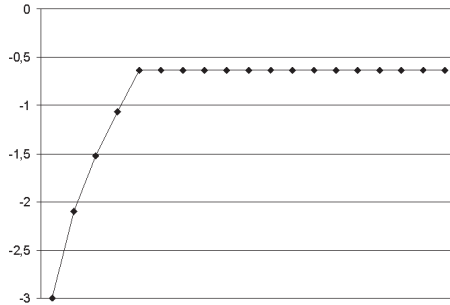


Fig. 10. Log-likelihood per sample of the models on the search path

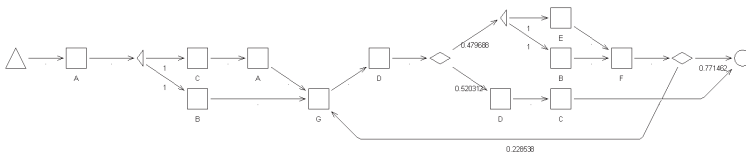


Fig. 11. Workflow 2: Result model using $LLH_{\min} = 0.1$

6 Summary and Future Work

We have presented a learning algorithm that is capable of inducing concurrent workflow models. This approach does not require unique activity nodes as other workflow induction algorithms do. We are convinced that the integration of workflow induction algorithms such as ours has the potential to provide a number of significant improvements to workflow management systems, including a shorter acquisition time for workflow models, higher quality workflow models with less errors and support for the detection of changing requirements.

Further work must be done to deal with noise, caused for example by erroneous workflow instances. Noise is especially critical if the dependency structure is affected. We are also working on algorithms that add explicit control flow constructs (Decision, Split and Join) to the induced model.

References

1. R. Agrawal, D. Gunopulos, and F. Leymann. Mining Process Models from Workflow Logs. In *Proc. of the sixth International Conference on Extending Database Technology (EDBT)*, 1998. [183](#), [189](#), [190](#), [192](#)
2. M. Berger, E. Ellmer, and D. Merkl. A Learning Component for Workflow Management Systems. In *Proceedings of the 31st Hawaii International Conference on System Sciences (HICSS-31)*, pages 754–763. IEEE, 1998. [190](#)
3. BOC. *ADONIS Version 3.0 - Users Guide*. BOC GmbH, Vienna, 1999. [184](#)
4. R. C. Carrasco and J. Oncina. Learning Stochastic Regular Grammars by Means of a State Merging Method. In *Second International Colloquium on Grammatical Inference and Applications (ICGI94)*, volume 862 of *Lecture Notes on Artificial Intelligence*, pages 139–152, Berlin, Germany, 1994. Springer-Verlag. [186](#)
5. J.E. Cook and A.L. Wolf. Automating Process Discovery through Event-Data Analysis. In *Proc. of the 17th International Conference on Software Engineering*, pages 73–82. Association for Computer Machinery, 1995. [183](#), [190](#)
6. J.E. Cook and A.L. Wolf. Event-Based Detection of Concurrency. Technical Report, CU-CS-860-98, Dep. of Computer Science, University of Colorado, 1998. [183](#), [190](#), [191](#), [192](#)
7. B. Curtis, M.I. Kellner, and J. Over. Process Modeling. *Communications of the ACM*, pages 75–90, September 1992. [184](#)
8. J. Herbst. Inducing Workflow Models from Workflow Instances. In *Proceedings of the 6th European Concurrent Engineering Conference*, pages 175–182. Society for Computer Simulation (SCS), 1999. [190](#)
9. J. Herbst and D. Karagiannis. Integrating Machine Learning and Workflow Management to Support Acquisition and Adaptation of Workflow Models. In *Proceedings of the Ninth International Workshop on Database and Expert Systems Applications*, pages 745–752. IEEE, 1998. [183](#), [185](#), [186](#), [188](#), [189](#)
10. H. Mannila, H. Toivonen, and A. I. Verkamo. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1(3):259–289, 1997. [190](#)
11. R. Parekh and V. Honavar. Automata Induction, Grammar Inference, and Language Acquisition. In Dale, Moisl, and Somers, editors, *Handbook of Natural Language Processing*. New York: Marcel Dekker, 1999. [186](#)
12. J.R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993. [185](#)
13. R. Srikant and R. Agrawal. Mining Sequential Patterns: Generalizations and Performance Improvements. In *Proc. of the Fifth International Conference on Extending Database Technology (EDBT)*, pages 3–17. Springer, 1996. [190](#)
14. A. Stolcke and S. Omohundro. Best-First Model Merging for Hidden Markov Model Induction. Technical Report, TR-94-003, Int. Computer Science Institute, 1994. [186](#)
15. C. Wargitsch. WorkBrain: Merging Organizational Memory and Workflow Management Systems. In *Workshop of Knowledge-Based Systems for Knowledge Management in Enterprises at the 21st annual German Conference on AI (KI-97)*, pages 214–219, Kaiserslautern, Germany, 1997. DFKI. [190](#)