Advances in Cognitive Systems 10 (2023) 53–70 Submitted 5/2020; published 5/2023

Learning Procedures by Augmenting Sequential Pattern Mining with Planning Knowledge

Melinda Gervasio MELINDA.GERVASIO@SRI.COM
Karen Myers KAREN.MYERS@SRI.COM
Artificial Intelligence Center, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025 USA

Abstract

Procedure automation can relieve users of the burden of repetitive, time-consuming, or complex procedures, and enable them to focus on more cognitively demanding tasks. Procedural learning is a method by which procedure automation can be achieved by intelligent computational assistants. This paper explores the use of filtering heuristics based on action models for automated planning to augment sequence-mining techniques. Sequential pattern-mining algorithms rely primarily on frequency of occurrence to identify patterns, leaving them susceptible to discovering patterns that make little sense from a psychological perspective. In contrast, humans are able to form models of procedures from small numbers of observations, even without explicit instruction. We posit that people can do so because of background knowledge about actions and procedures, which lets them effectively filter out incoherent or impractical sequential patterns. The action models foundational to artificial intelligence planning provide semantics for actions, supporting the design of heuristics for eliminating spurious patterns discovered from event logs. We present experiments with various filters derived from these action models, the results of which show that filters in greatly reduce the number of sequential patterns discovered without sacrificing the number of correct patterns found, even with small, noisy event logs.

1. Introduction

Humans are adept at learning procedures from observation. Children watch and learn from parents and teachers, siblings and playmates. New hires shadow experienced professionals to learn about the organization's standard procedures. An apprentice learns new skills by watching a master at work. Although interaction and direct teaching typically accompany learning from observation, we can recognize meaningful patterns in observed behavior even without explicit demonstration or instruction. And we can do so without requiring large numbers of examples. We can identify what is relevant and what is not, so that even with just a few cases of some unknown procedure being executed, we can learn the underlying process.

In this work, we set out to address the task of discovering automatable procedures from transaction logs of computer applications. The task involves a data set that, in general, contains multiple instances of a procedure, with not every procedure present in every log. Furthermore, each log may include zero or more such procedures, with the steps of different procedures potentially being interleaved. Logs may contain extraneous actions and some actions may not be amenable to automation. Given a data set of such logs, can we use learning from observation to identify and extract the automatable procedures?

The problem of learning procedures from examples has been tackled through macro-operator learning in the AI planning community. Some work (Fikes et al., 1972; Minton, 1985; Mooney, 1988) assumed the availability of examples of the target plan, with the task being to generalize those examples. In our setting, such examples are only implicit in the data and one challenge is to discover the examples. Other work extracted macros directly from problem-solving traces (Korf, 1985; Iba, 1989), which would be analogous to the transactions in our setting. Given perfect knowledge of the domain actions and their preconditions and effects, and a state evaluation function indicating the desirability of a state to help identify possible subgoals, one might apply similar techniques to extract candidate macros from a training set of transactions, and then use statistical methods on a validation set to identify the high-utility ones. However, although we may have some domain knowledge in our setting, such domain knowledge is not guaranteed to be complete and correct, leaving this approach susceptible to overlooking potentially useful procedures.

The inverse of this knowledge-driven approach is unsupervised discovery of procedures from transaction data. The discovery of sequential patterns in particular has been tackled through sequence mining, a popular computational technique for discovering patterns within records of larger sequences. It has been used to identify consumer purchasing behaviors in transaction databases (Agrawal & Srikant, 1995), as well as motifs (gene and protein sequences with distinct functions) within genomic strings (Abouelhoda & Ghanem, 2010). Process mining, another technique, has been used to discover process knowledge from event logs (van der Aalst et al., 2012). However, these are all designed to operate on large volumes of data and rely on frequency as the main indicator of patterns. They find frequent patterns that signify general trends and, in many cases, the 'correctness' of patterns is immaterial: they are often intended simply as starting points for a human expert to analyze or refine.

In contrast, we wish to discover patterns that enable useful automation, which must correspond to meaningful, coherent sequences of actions. Unlike the characters that comprise protein sequences or the events in transaction logs, action sequences are also rich in structure, with parameters, preconditions and effects, inputs and outputs. Plans similarly involve semantically rich actions with formal specifications that let reasoning determine their applicability and effects in a given state, enabling both the construction of plans (Ghallab et al., 2004) and composition into macro-operators (Fikes et al., 1972; Minton, 1985; Korf, 1985; Mooney, 1988; Iba, 1989). In this paper, we investigate whether one can augment sequence-mining techniques with filtering heuristics derived from planning knowledge to identify promising candidate sequences. Most domains lend themselves naturally to the formulation of rules that capture information about what makes a good candidate sequence. For procedures, enabling relationships (e.g., conditions that should hold for an action to be performed) are a natural focus, as they capture the underlying 'physics' of the domain. We use action models to represent these relationships, from which we devise two sets of heuristics: action filters to eliminate noise from discovered patterns and candidate filters to eliminate undesirable patterns. We also report experimental results that show the ability of these filters to improve precision without sacrificing recall.

We begin by defining the sequential pattern-mining problem and our approach to discovering frequent parameterized action sequences. We then present the results from a baseline study, illustrating the problem with using simple frequency-based techniques to find candidate procedures—a problem exacerbated by the presence of noise. After this, we introduce action models and the filtering heuristics designed to eliminate undesirable candidates. We next present our experiments with filters on both noise-free and noisy data sets, which show the effectiveness of filters in increasing precision with minimal decrease in recall. The work is a first attempt to use planning knowledge

Table 1. Sequential pattern mining is limited to finding patterns over actions (without any arguments) (*Result 1*). Simply appending arguments to actions (*Result 2*) does not enable the parameter generalization required to achieve the *Desired Result*. (*Support* refers to the number of times that the pattern appears in the input sequences.)

Event Logs	Result 1	Result 2	Desired Result
Get(John,ID45) Approve(ID45)	*Sequence 1, Support 3*: Get Approve	*Sequence 1, Support 1*: GetJohnID45 ApproveID45	*Sequence 1, Support 2*: Get(Name1,Id2) Approve(Id2))
Get(Jane,ID21) Approve(ID62)		*Sequence 2, Support 1*: GetJaneID21 ApproveID62	*Sequence 2, Support 1*: Get(Name1,Id2) Approve(Id3)
Get(Jill,ID37) Approve(ID37)		*Sequence 3, Support 1*: GetJillID37 ApproveID37	

to improve sequence mining for procedure learning, so we discuss its contributions in the context of related research in procedural learning and sequence mining. We conclude with a discussion of plans for future work in this area.

2. Sequential Pattern Mining for Procedure Learning

Sequential pattern mining is a specialized data-mining task for finding sequential patterns in data (Chand et al., 2013; Fournier-Viger et al., 2017). It has been applied to problems like market basket analysis, biological sequence discovery, clickstream analysis, and workflow verification. However, sequence-mining algorithms almost always rely purely on frequency of occurrence to identify candidate patterns. In particular, all such algorithms specify a *minimum support* parameter that denotes the minimum frequency of occurrence for a pattern to be considered a viable candidate. And while some algorithms accommodate additional constraints (Negrevergne & Guns, 2015; Pei et al., 2004; Pei & Wang, 2002), they are still limited to looking at relatively shallow properties rather than knowledge about what makes a good pattern.

Sequence-mining algorithms today are robust and highly efficient, but they have two main limitations when applied to procedural learning. First, they operate over sequences of atoms, such as DNA sequences, items purchased, URL clickstreams, and event logs. In contrast, procedure mining should handle input sequences that are composed of *parameterized actions* and discover relationships between those parameters to achieve generalization. Second, there are no semantics associated with these atoms, so they are often simply converted into integers for compactness. Because sequence-mining algorithms are purely statistical, the atoms over which they operate have no semantics. In contrast, the actions in procedures have meaning: they are intended to achieve something, they have preconditions and effects, and they manipulate data.

Here we first review how to address the first limitation using a technique described by Gervasio and Lee (2013) for learning *action idioms*, i.e., groups of low-level actions from instrumentation logs abstracted to human-level actions. The work described in the remainder of the paper addresses the second limitation. Consider the example in Table 1. *Event Logs* shows three event sequences,

each composed of a *Get* action followed by an *Approve* action. *Get* is an ID retrieval with one input parameter (the name of the person whose ID to retrieve) and one output parameter (the person's ID). *Approve* is an approval action that takes one input (the ID of the person to approve). The first and third input sequences are examples of the same procedure, involving the ID retrieval and subsequent approval for the same person. The second sequence involves a different procedure that retrieves one person's ID and approves another.

A straight application of sequence mining would look only at the action names (*Get* and *Approve*) and discover a single pattern covering all three sequences (Table 1, *Result 1*). A possible approach to including action arguments is to translate each action in the input sequence into a new string that concatenates the action name with its arguments. However, this leads to the opposite problem of undergeneralization, with each input sequence being recognized as a different pattern (Table 1, *Result 2*). The desired result is one that distinguishes between the patterns by recognizing the relationships between the action arguments as well (Table 1, *Desired Result*).

To achieve the desired parameter generalization, Gervasio and Lee's (2013) method applies sequence mining on the actions only (Table 1, *Result 1*). It then applies a postprocessing step to partition the supporting sequences according to parameter matches by going through each supporting sequence and assigning unique ids to the different argument values in order. This enables the recognition that there are two unique argument values in the first and third sequences but three in the second. It also reveals that the second unique argument value in the first and third sequences is the second argument for the first action and the sole argument for the second. For list and set (collection) arguments, unification and variablization can be extended to find supports from collections to individuals (e.g., $first([a,b,c]) \rightarrow a$) and from individuals to collections (e.g., $list(a,b,c) = [a,b,c]$) (Eker et al., 2009). This modified sequential pattern-mining approach serves as the basic method we use in the study described next.

3. Baseline Study

To verify that this approach to finding action idioms can also discover parameterized sequential patterns, we applied it to a data set of transactional logs. The purpose of the study was to provide baseline results against which to evaluate the augmented method that uses filtering heuristics.

3.1 Baseline Data Sets

To evaluate this basic approach, we repurposed transactional log data sets collected by the IEEE Task Force on Process Mining available through the 4TU.Centre for Research Data (2016). Specifically, we used the Large Bank Transaction Process data set (Muñoz-Gama, 2014), a collection of synthetic event logs generated from a model of bank transactions. The IEEE Task Force collection includes both natural and synthetic logs. We chose to work with the synthetic logs because they included the Petri net model from which the logs were generated, letting us develop 'ground truth' (i.e., the target procedures for our mining algorithm) to use for evaluation. Figure 1 shows the complete bank transactions model and Figure 2 shows the portion corresponding to sender (customer) authentication: we focused on this portion in our experiments, extracting 28 target (ground truth) procedures, corresponding to the paths depicted in Figure 2.

Like most transactional logs, each entry in the bank transaction data set consists simply of a log identifier and an event type. To transform the data into a form more akin to the parameterized action logs that are our focus, we modeled each action (square) in Figure 2 in terms of its inputs

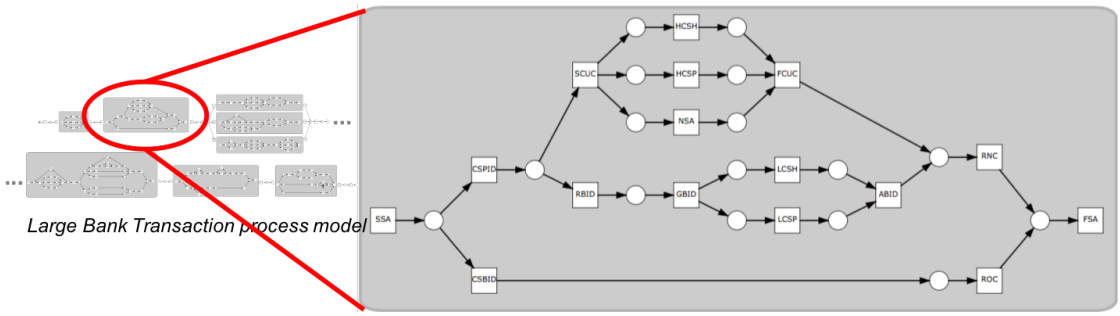

Figure 1. Large Bank Transaction Petri Net Model. The callout shows the Sender (Customer) Authentication subprocess used in the experiments discussed in this paper. The complete model on the left is shown only to provide context; its details are not relevant here.

and outputs—i.e., the information it consumes and produces. We then translated each event in the data set to its equivalent action to generate the log we used in our experiments. Table 2 illustrates this transformation. The data set provided a few different synthetically generated logs. For our experiments, we used the one comprising 2000 noise-free logs of observed transaction sequences (corresponding to 2000 different customers). However, because we expect to have many fewer logs in our target application, we mined procedures from only 100 randomly selected logs. We will refer to this as the *noise-free* data set.

Because real-world transactional logs are unlikely to be completely devoid of noise, we also created noisy versions of the data to investigate its effects. To create the noisy versions, we injected two types of noise: redundant actions and extraneous ones. We also considered removing actions but decided that this did not make sense in the procedural automation setting because traces with missing actions would not have accomplished the desired effects and thus are not really examples

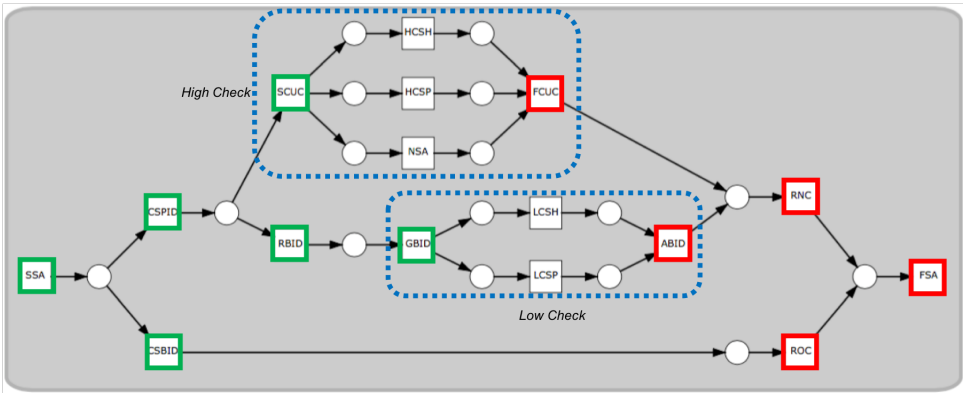

Figure 2. The target (ground truth) procedures in the *Sender Authentication* subprocess consist of all possible paths between a green square and a red square. The diagram is a Petri net, so all branches emanating from a square can be traversed in any order. This leads to six different sequences for the *High Check* subgraph and two for the *Low Check* subgraph.

Table 2. An example conversion from an event log to an action log. Each atomic event is translated into its equivalent parameterized action.

Atomic Event	Parameterization
trace_0 SSA	StartAuthentication(+Person1, -AuthId2)
trace_0 CSPID	CheckPersonalID(+Person1, +AuthId2)
trace_0 RBID	RequestBankID(+Person1,+AuthId2, -RequestId3)
trace_0 GBID	GenerateBankID(+Person1,+RequestId3, -BankId40)
trace_0 LCSP	LowCheck(+Person1, +BankId4, PROFILE)
trace_0 ABID	ActivateBankID(+Person1, +BankId4)

of any target procedure. We injected redundant actions by repeating each action in the log with 0.1 probability. Similarly, we inserted the extraneous actions by going through each action in a log and inserting a randomly selected action with 0.1 probability. We created three noisy data sets: one with *redundant* actions only, one with *extraneous* actions only, and one with *both* redundant and extraneous actions.[1]

3.2 Baseline Results

We were interested in how well the candidate sequences discovered by sequence mining would match ground truth, so we measured *precision* and *recall*. We expected recall to be high, since sequential pattern mining is designed to discover as many patterns as possible. But because every subsequence of a frequent sequence will also be a frequent sequence, we expected it to find many spurious procedures and thus precision to be low. This subsumption property has led to much work in frequent pattern mining to focus on finding only maximal sequences (i.e., frequent sequences that are not contained in a longer frequent sequence) or closed sequences (i.e., maximal sequences that are not contained in a longer one with the same support). For procedure mining, however, we do not want to limit ourselves to closed or maximal sequences because subsequences are likely to be useful as well, as seen in the ground truth procedures.

To establish a baseline, we ran the CM-SPAM[2] algorithm (Fournier-Viger et al., 2014) over the transformed action log with a minimum support of five. Table 3 summarizes the results for the *Noise-free* case and the three noisy data sets (*Redundant*, *Extraneous*, *Both*). As expected, recall is high (perfect in the Noise-free case) but precision is fairly poor (12%). Performance degrades with the noisy data sets, with more incorrect candidates (lower precision) found in all three, and fewer correct candidates (lower recall) found in the *Extraneous* and *Both* noisy data sets.

[1] We also tried both smaller and larger probability values. These resulted in the expected decrease and increase, respectively, in the number of candidates generated but the overall results for the later studies using filters did not change.

[2] Sequence-mining algorithms target very large data sets, so they are designed to be highly efficient for particular problems. Our objective was to see whether we could leverage established techniques to learn automatable procedures from small amounts of data. We decided to use the CM-SPAM implementation in the Sequential Pattern Mining Framework (SPMF) library (Fournier-Viger et al., 2016) because it was particularly efficient, provided all frequent sequences, and had a tunable maximum gap parameter.

Table 3. Baseline performance. Sequence mining finds all true candidates (perfect recall) but many false ones (low precision). All types of noise increase the number of incorrect candidates found, while extraneous actions and both redundant and extraneous actions also decrease this measure.

Data set	Cands	True	Recall	Prec
Noise-free	229	28	1.0000	0.1223
Redundant	289	28	1.0000	0.0720
Extraneous	252	25	0.8929	0.0992
Both	372	19	0.6786	0.0511

4. Knowledge-Guided Candidate Filtering

Sequential pattern mining relies solely on frequency of occurrence to identify candidates. While this may be sufficient for finding repetitive occurrences such as buying patterns or frequent clickthrough behavior, it is inadequate for finding procedures—i.e., meaningful action sequences intended to achieve some goal. Action sequences based purely on frequency of observation may not always be good candidates for automation. As seen in the baseline study, they may contain (non-observable) intervening actions, systematic noise, and non-automatable actions. Extraneous actions that serve no purpose cannot be filtered out, nor can nonsensical action sequences such as those that start with actions whose preconditions have not been established. The action models used in AI planning are designed precisely to enable reasoning about such dependencies. With this in mind, we set out to see how we could use action models to filter out the false candidates from many candidates generated by sequential pattern-mining algorithms. Unlike in AI planning, our approach does not require a complete and correct set of action models since we only use the action models to devise filtering heuristics and not to solve planning problems. Although we should get better performance with complete and correct models, even imperfect action models should reduce the noise in the candidate sequential patterns. And in our domains of interest, this partial information is readily available.

4.1 Action Model

We decided to use a hybrid notation for action models that combines standard STRIPS-style semantics, which capture preconditions and effects, with a dataflow-oriented representation of actions in terms of their inputs and outputs.

4.1.1 Planning Domain Definition Language.

The Planning Domain Definition Language (PDDL) (Kovacs, 2011; McDermott et al., 1998), a descendant of STRIPS (Fikes & Nilsson, 1971), is the de facto standard for encoding first-principles planning knowledge. In PDDL, the domain description includes a model for each action, which comprises a set of (typed) parameters, a set of preconditions that specify when it can be applied, and a set of effects that define the results of its execution. For example, Table 4 (left) shows the action definition for a *FinishAuthentication* action in the Bank Transaction domain. It has two parameters (the sender and the authentication ID), requires that the authentication be in process and that the customer already have been registered, and results in the sender being authenticated.

In our work on learning procedures from demonstration in informational domains (Gervasio & Murdock, 2009; Eker et al., 2009; Garvey et al., 2009), we found data flow—i.e., the information

Table 4. PDDL specification of the *FinishAuthentication* action (left) and dataflow-based specification of *StartAuthentication* action (right).

PDDL Specification

```
(:action FinishAuthentication
  :parameters (?sender ?aid)
  :precondition
    (and (authenticating ?sender ?aid))
         (registered ?sender))
  :effect
    (and (not authenticating ?sender ?aid))
         (authenticated ?sender)))
```

Dataflow-Based Specification

```
<action id="StartAuthentication">
  <description>
    Start authentication
  </description>
  <inputParam id="sender">
    <description>
      the sender
    </description>
    <typeRef typeId="string"/>
  </inputParam>
  <outputParam id="aid">
    <description>
      authentication id
    </description>
    <typeRef typeId="string"/>
  </outputParam>
</action>
```

producer-consumer relationship between actions—to be a particularly useful concept, with most actions involving the production of information required by subsequent actions and/or the consumption of data generated by previous actions. By focusing on the identification and generalization of data flow, we were able to develop techniques for learning general, parameterized procedures from as little as one example. To support the reasoning required to identify and generalize data flow, we devised a representation in which actions have typed parameters, each designated as an input or an output, with the semantics that, given particular input arguments, executing the action will generate the output arguments. For example, Table 4 (right) shows the action definition for the *StartAuthentication* action: Given as input a sender (name), it outputs a new authentication ID.

We extended this dataflow-oriented action model for task learning with preconditions and effects, as in PDDL, to encode the planning knowledge for a domain. We note that classic work on learning plans from examples assumes information not just about the actions that were executed but also the state of the world before and after the execution of each action (Fikes et al., 1972; Minton, 1985). However, action or transactional logs typically lack such state information. And yet humans can look at such action sequences and infer the intervening states based on their knowledge of the actions. They can also identify related actions and ignore irrelevant ones. This was the primary motivation behind our investigation into the use of heuristics based on action models to filter the candidate patterns discovered through sequential pattern mining.

4.2 Candidate Filters

The preconditions and effects and the inputs and outputs of an action provide valuable information regarding whether inclusion of the action in an observed sequence makes sense. To leverage this information in identifying good candidates for automation, we developed a set of filtering heuristics based on common-sense knowledge about the nature of actions and procedures in this domain. We note that these filters, summarized in Table 5, are intended to serve as examples and to evaluate the idea of knowledge-based filtering; they are not meant to be complete or definitive.

Table 5. Heuristics for filtering candidate sequential patterns based on common-sense knowledge about actions and procedures in the banking domain.

Filter	Description
Precondition	Discards candidates with any action having an unsatisfied or unsatisfiable precondition
Start	Discards candidates that do not begin with an action that starts a process
Finalize	Discards candidates that do not end with an action that completes a process
Complete	Discards candidates that have an action that starts (ends) a process without a matching action that ends (starts) it
Branch	Discards candidates that do not start with a recognized branching action

The *Precondition* filter discards any candidate containing an action whose preconditions are not satisfied. This is because a procedure cannot be executed to completion if any of its actions has an unsatisfied precondition. Since event logs do not contain state information, we rely on information about acceptable initial state conditions instead. To signify that these conditions denote possible states from which an authentication procedure might be initiated, we augment the action model with metadata identifying such conditions. For example, in this domain, the condition of a customer being known is a valid (possible) initial condition but the condition of authority being notified is not.

The next three filters rely on knowledge about the processes in the domain, as tracked by the conditions established by the actions. Specifically, processes must start and they must end. By designating certain conditions as referring to a process, we can recognize when an action starts the process (i.e., establishes the condition) or finishes it (i.e., negates the condition). For example, the action *StartAuthentication* has the effect of *(authenticating ?customer)* (that is, starting an authentication process), while the action *FinishAuthentication* has the effect of *(not (authenticating ?customer))* (i.e., finishing it). The *Start* filter requires candidates to begin with an action that starts a process, while the *Finalize* filter requires them to end with an action that finishes a process. The *Complete* filter combines the two constraints, requiring that every action that starts a process have a corresponding action that ends it.

The final filter, *Branch*, discards candidates that do not begin with an action recognized to be one of the options. This may not apply to procedures in some domains, but for banking transactions, we know there are often steps that can be executed in any order—i.e., any of the steps can start a procedure provided the other steps are carried out later. In the banking transaction model, for example, the check for high-risk (i.e., new or unknown) clients involves checking the customer's banking history, checking the customer's profile, and notifying authorities. The intuition behind the *Branch* filter is that action sequences beginning with any of these branching actions are likely to correspond to good candidates.

In addition to the candidate filters in Table 5, we also devised action filters for discarding unnecessary actions in candidates, as summarized in Table 6 (a). The *Contribution* filter requires that every action serve a purpose—i.e., it either establishes a condition or produces an output required by a subsequent action or it requires a condition or input produced by a preceding action. The *Duplicate* filter discards actions that are exact repetitions of the previous actions, the rationale being that repeating the same action serves no purpose. We designed these action filters to eliminate

Table 6. (a) Heuristics for filtering unnecessary actions from candidate sequential patterns; (b) Combinations of heuristics for filtering candidate sequential patterns.

Filter	Description
Contribution	Discards actions that are not required by any succeeding action or that do not require any preceding action
Duplicate	Discards immediate repeated actions in a sequence
Gap	Ignores up to a certain number of intervening actions
Combo1	Discards actions that do not pass the Contribution filter and then candidates that do not pass the Precondition filter and at least two of Branch, Start, Finalize, and Complete
Combo2	Discards actions that do not pass the Contribution filter and then candidates that do not pass the Precondition filter and either the Branch or Complete filter

extraneous actions, whether due to inadvertent execution, multitasking, or some other reason, and that may lead to spurious patterns being discovered. Meanwhile, the *Gap* filter uses the maximum gap constraint in pattern-mining algorithms such as CM-SPAM, which is designed to accommodate noise by allowing gaps up to a certain length within a sequence. For the experiments described below, we allowed up to one intervening action between elements of a candidate sequence, which we implemented by setting the CM-SPAM maximum gap parameter accordingly.

The *Precondition* and *Contribution* filters each check for *necessary* conditions, while the rest check for *desirable* ones. To determine whether first eliminating invalid sequences would improve the precision of the heuristics, we also devised two composite filters that require passing both the *Contribution* and *Precondition* filters and at least one of the other candidate filters. Table 6 (b) describes these combinations.

5. Experiments with Filters

To evaluate our approach of augmenting sequential pattern mining with filtering heuristics derived from action models, we conducted experiments using the data sets created for the baseline study. As in that study, we ran the CM-SPAM algorithm augmented with parameter unification and variabilization to generate the candidate sequences, and then applied our filtering heuristics to eliminate undesirable candidates. The one exception to this post-processing approach to filtering was the *Gap* filter, which we implemented using the maximum gap constraint of the CM-SPAM algorithm instead. Our main hypothesis was that the filters would eliminate bad candidates from consideration, thereby improving precision over baseline results. However, we were also interested in whether recall would suffer and by how much.

5.1 Experimental Results

Table 7 summarizes the results with the different filters, alone and in combination, applied over the frequent sequences found by CM-SPAM on the noise-free data set. The individual candidate filters never degrade precision and, in most cases, improve it, although recall, unsurprisingly, is sometimes affected. Among the individual filters, the *Branch* filter results in the highest precision, but

Table 7. Results of filtering on noise-free data set showing the number of candidates found, the number of correct candidates among those, recall, and precision.

Filter	Cands	True	Recall	Prec
Precondition	229	28	1.0000	0.1223
Start	94	17	0.6071	0.1809
Finalize	87	15	0.5357	0.1724
Complete	156	28	1.0000	0.1795
Branch	92	17	0.6071	0.1848
Contribution	215	28	1.0000	0.1302
Duplicate	229	28	1.0000	0.1223
Gap	1020	28	1.0000	0.0275
Combo1	128	26	0.9286	0.2031
Combo2	169	28	1.0000	0.1657

it also has markedly lower recall. The best performance overall is achieved by the *Combo1* filter, which records the highest precision and almost perfect recall.

The *Contribution*, *Duplicate*, and *Gap* action filters were designed primarily to address noise and so we did not expect them to help much in the noise-free case. Because the noise-free data set contained no repeated actions, the *Duplicate* filter offers no improvement over the baseline. The *Gap* filter, because it is designed to recognize patterns even with segments that do not match (allowable gaps), results in an explosion in the number of patterns found, greatly decreasing precision. The *Contribution* filter has a mild positive effect on performance, removing a small number of false positives for a slight improvement in precision.

Table 8 summarizes the results of applying the different filters on the noisy data set with only *Redundant* actions. With the exception of the *Gap* filter, every filter improved precision. Not surprisingly, the *Duplicate* filter, which removes repeated actions (i.e., exactly the injected noise), resulted in the best precision on the *Redundant* data set, matching the performance of the baseline in the noise-free case. And applying the *Duplicate* filter in combination with any other filters

Table 8. Results of filtering on noisy data set with *Redundant* actions.

Filter	Cands	True	Recall	Prec
Precondition	251	28	1.0000	0.1116
Start	156	17	0.6071	0.1090
Finalize	134	15	0.5357	0.1119
Complete	272	28	1.0000	0.1029
Branch	149	17	0.6071	0.1141
Contribution	357	28	1.000	0.0784
Duplicate	229	28	1.000	0.1223
Gap	2201	28	1.0000	0.0127
Combo1	128	26	0.9286	0.2031
Combo2	169	28	1.0000	0.1657

Table 9. Results of filtering on noisy data set with *Extraneous* actions.

Filter	Cands	True	Recall	Prec
Precondition	232	25	0.8929	0.1078
Start	99	15	0.5357	0.1515
Finalize	89	13	0.4643	0.1461
Complete	159	25	0.8929	0.1572
Branch	98	15	0.5357	0.1531
Contribution	208	25	0.8929	0.1202
Duplicate	251	25	0.8929	0.0996
Gap	1266	28	1.0000	0.0221
Combo1	116	26	0.8214	0.1983
Combo2	159	28	0.8929	0.1572

matched performance in the corresponding noise-free case (results omitted for brevity). The best performance, however, results from the combination filters.

The *Gap* filter, as in the noise-free case, substantially increased the number of candidates, resulting in greatly degraded precision. The *Gap* filter is designed to find candidates that would not otherwise be found because of insufficient support due to noise. For example, if there are three sequences *ACB*, *AB*, and *ADB*, then an allowable gap of one would find the sequence *AB* with support three, while allowing no gap would not find any sequence with support greater than one. Thus, the result here may be explained by the fact that, without filtering, all ground truth candidates are already found—i.e., there is nothing useful left for the *Gap* filter to find. Instead it detects many other extra candidates. We had expected the *Gap* filter to have some positive effect on small data sets having patterns that have frequencies of occurrence close to the minimum, but the extremely high number of other patterns found makes it unlikely to be useful. Table 9 shows the results for the different filters on the noisy data set with *Extraneous* actions only and Table 10 shows the results for the noisy data set with *Both* redundant and extraneous actions. The results are similar to those for noise due to only *Redundant* actions. Again, with the exception of the *Gap* filter, all the filters aid performance, with the combination filters resulting in the greatest improvement.

5.2 Discussion

The experimental results support our conjecture that standard sequential pattern-mining techniques discover many irrelevant action sequences and that filters based on action models can eliminate most of them. Sequential pattern mining relies almost exclusively on frequency of occurrence to identify patterns. This is often sufficient for applications such as consumer product recommenddations, motif detection, or compliance checking, where the cost of an incorrect pattern is small and thus there is generally a greater emphasis on recall. Often, there is also a human to ultimately assess the quality of the discovered patterns: a consumer deciding whether to accept a product recommendation, a scientist verifying a motif, or a business ensuring best practices. However, with patterns that are intended for automation, it is important to find correct procedures and thus there is a need for greater precision.

Inspired by the action models used in AI planning to reason about how actions in a plan establish preconditions or generate the inputs for subsequent actions, we set out to develop heuristic

Table 10. Results of filtering on noisy data set with both *Redundant* and *Extraneous* actions.

Filter	Cands	True	Recall	Prec
Precondition	218	19	0.6786	0.0872
Start	135	11	0.3929	0.0815
Finalize	131	9	0.3214	0.0687
Complete	252	19	0.6786	0.0754
Branch	134	12	0.4286	0.0896
Contribution	296	19	0.6786	0.0642
Duplicate	268	26	0.9286	0.0970
Gap	1339	28	1.0000	0.0209
Combo1	108	17	0.6071	0.1574
Combo2	149	19	0.6786	0.1275

filters that identify coherent candidate action sequences. The idea was to encode in these filters the knowledge of which patterns 'made sense' from a procedural perspective—i.e., patterns whose actions were related in some way or were otherwise indicative of a procedure. Our experiments showed that filters derived from action models can successfully find coherent action sequences suitable for automation. Every candidate filter and all but the *Gap* action filter eliminated undesirable candidates and most did so without sacrificing recall. Baseline precision, based on standard sequential pattern-mining techniques, is just over 12% in the noise-free case and ranges from 5 to 10% for the noisy data sets. With the filters, however, precision can reach up to almost 20% in the noise-free case and up to 15 to 20% for the noisy data sets. This is a substantial 100–200% improvement in performance. It remains to be seen whether a one-in-five hit rate for discovered procedures suffices for automation is sufficient. Nevertheless, these results show that knowledge-based heuristics for filtering candidates can be an effective addition to standard sequential pattern mining and guide discovery toward meaningful action sequences that correspond to useful, automatable procedures.

6. Related Work

Procedural learning—the acquisition of skills for performing tasks—has been well studied in the cognitive science and AI communities. A broad array of approaches has been explored, including learning from problem solving (Laird et al., 1986), observation (van Lent & Rosenbloom, 2001), instruction (Blythe, 2005), multiple modalities (Allen et al., 2007), demonstration (Gervasio & Murdock, 2009), and solution traces (Li et al., 2009).

In the AI planning community, the concept of macro-operators was conceived as a means to improve the efficiency of planning by compiling the search to determine a sequence of actions to achieve a goal from a given initial state. Some approaches to learning macro-operators attempt to extract them from available solutions (Korf, 1985; Mooney, 1988), while others analyze problem-solving traces generated during search for solutions. These rules can be learned from problem-solving traces (Minton et al., 1989; Gratch & DeJong, 1991) or through static analysis of problem space definitions (Etzioni, 1993). The procedure automation that drives our work on procedure discovery is similarly motivated by efficiency, but our setting involves finding procedures in logs

of actions executed by any number of people rather than the actions of the planning agent itself or intentional demonstrations by an expert. Furthermore, the work on learning macro-operators and search control rules relies on having complete and correct domain knowledge to ensure the correct-by-construction plans that underpin learning. Our approach instead utilizes the available domain knowledge to identify promising patterns from candidates obtained through unsupervised methods.

Procedures are a type of sequential pattern and the idea of finding sequential patterns in data has been explored in a number of fields. In the data mining community, Agrawal and Srikant (1995) introduced sequential pattern mining in their seminal work on market basket analysis, which can be used to drive decisions about marketing activities, such as campaigns to recommend products, based on discovered patterns in consumer behavior. Pattern-mining methods developed since then (e.g., Fournier-Viger et al., 2014; Pei et al., 2004; Srikant & Agrawal, 1996; Zaki, 2001) differ in how they search the space of patterns, how they represent the database, how they generate next candidates, and how they determine support (frequency of occurrence) for patterns (Fournier-Viger et al., 2017). Because the applications driving the work in the data mining community involve very large databases, research in this area has focused primarily on time and space efficiency.

Sequence mining in bioinformatics (Abouelhoda & Ghanem, 2010) also involves large data sets and emphasizes efficient algorithms. In contrast to data-mining tasks, which involve finding patterns in large numbers of relatively short sequences over many possible items (alphabets), biological sequence mining finds patterns in very long sequences for small alphabets. Furthermore, the main driver for biological mining is finding repeated strings that correspond to some significant biological structure or function, such as motifs (e.g., Bailey et al., 2009; Chou & Schwartz, 2011).

Our particular interest is in discovering repeated action sequences that correspond to execution traces of processes or procedures that could be automated. This is closest to process mining, which attempts to discover process knowledge from event logs (van der Aalst et al., 2012). Process mining has been applied to a variety of domains, including health care (Rojas et al., 2016), software development (Cook & Wolf, 1998), public works infrastructure (van der Aalst et al., 2007), and other business settings. Much work in this area is concerned with finding processes to support analysis, such as conformance checking and workflow enhancement. Because real-life processes can be quite complex, process mining is designed to discover control structures like loops and conditionals, with Petri nets being a popular representation for learned models. However, consideration of other attributes such as actors, time stamps, and resources is typically done outside mining itself.

Constraint-based mining (Negrevergne & Guns, 2015; Pei et al., 2004; Pei & Wang, 2002) provides an avenue for biasing the search for frequent patterns by requiring that they satisfy user-specified constraints. However, the constraints have typically been limited in scope and focused on syntactic features of sequential patterns. Negrevergne and Guns categorize them into constraints on patterns (e.g., minimum size), constraints on cover sets (e.g., minimum frequency), constraints on inclusion relations (e.g., maximum gap), and preferences over candidate patterns (e.g., maximal patterns). Most work on sequence mining develops specialized algorithms for a subset of them. Our task requires extensions to semantic constraints on action sequences of task-oriented procedures. The patterns we wish to discover are intended for automation and must thus correspond to meaningful, coherent sequences of actions. Unlike the characters that comprise protein sequences or the simple events in transaction logs, actions are rich in structure. They have parameters (often typed), preconditions, and effects, they take inputs and generate outputs, and they are organized in hierarchies. Furthermore, in our setting, we cannot assume voluminous action logs from which to discover patterns. This greatly lowers the tolerance for noise and increases the need for effective generalization from small numbers of examples.

7. Conclusions and Future Work

In this paper, we proposed a hybrid approach to procedure mining that combines knowledge-based heuristics derived from AI planning models with statistical techniques from sequential pattern mining to discover candidate action sequences for automation. This leverages efficient sequence-mining techniques to find frequent action sequences from logs of user actions that serve as candidate action sequences for automation. Using action models that provide a semantic representation of actions in terms of their preconditions and effects, as well as their inputs and outputs, we devised filtering heuristics to help identify good candidate sequences for automation. We conducted several experiments to evaluate the usefulness of filters on both noise-free and noisy data sets. The results showed that the filtering heuristics based on action models eliminate many irrelevant sequences discovered by standard sequential pattern mining.

Our approach introduces an additional cost to deployment above and beyond that of purely statistical methods in that it requires the formulation of action models for any new domain to which it is applied. However, this is a one-time expense that could be justified for many applications if our preliminary results on improved quality of recognition hold more generally. As discussed previously, the filtering heuristics presented in this paper may not apply to all domains. Some, like the *Branch* filter, take advantage of characteristics specific to banking transactions. On the other hand, we are likely to be able to leverage characteristics specific to any domain by designing new filters that encode the knowledge about its processes. For example, in a customer support call-center application, all procedures might begin with creating an incident report and filling in the date and time of the call. They may also include retrieving resolution options based on a standard operating manual or similar incidents, and they may all conclude with a resolution or an escalation.

Some applications may require learning from even smaller numbers of examples. These will need heuristics like the *Gap* filter to enable finding sequences for which there is not enough support otherwise. For example, an abstraction filter that recognizes some actions are variants of others because they achieve the same cumulative effects would allow sequences that differed only in the variant used to be grouped. Prior work on learning disjunctive macro-operators (Shell & Carbonell, 1989) may provide other techniques for generalizing structure and learning compact procedures.

We believe that our use of action models to inform statistical sequence mining has potential benefits that go beyond increased precision. One such benefit is providing rationale for mined sequences. The preconditions and effects characterize the causal structure of the procedure: what it does, when it can be done (its accumulated preconditions), and why it would be done (its accumulated effects). We might use this information to generate explanations for a user to accompany suggestions for task automation, drawing on explanatory techniques such as those described in Seegebarth et al. (2012).

Our work has been motivated by the ultimate objective of enabling process automation for real-world usage. The focus to date has been on a collaborative task management tool called Task Assistant (Peintner et al., 2009). This system supports distributed human teams in collectively executing complex coordinated processes (e.g., Standard Operating Procedures) through an explicit representation of tasks, dependencies, deadlines, and status. Task Assistant has been deployed successfully to a number of operational user communities, including the U.S. Pacific Fleet (PACFLT), the U.S. Strategic Command (STRATCOM), and the Kansas National Guard. With those deployments, we have seen the opportunity to improve team and individual efficiency by introducing automation of frequently executed support tasks, many of which focus on information retrieval to aid human decision making. Our first approach to procedure automation for Task Assistant relied on learning from demonstration (Myers et al., 2011), which required substantial

user effort to recognize the need for automation and to explicitly walk through the procedures. Our engagement with the user community has shown a strong desire for automating procedures with little human intervention, as would be enabled by the procedure-mining technique in this paper.

Acknowledgements

This material is based upon work supported by the Office of Naval Research (ONR) under Contract N00014-15-C-5040. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of ONR.

References

Abouelhoda, M., & Ghanem, M. (2010). String mining in bioinformatics. In M. M. Gaber (Ed.), *Scientific data mining and knowledge discovery*: *Principles and foundations*, 207–247, Berlin, Germany: Springer-Verlag.

Agrawal, R., & Srikant, R. (1995). Mining sequential patterns. *Proceedings of the Eleventh International Conference on Data Engineering* (pp. 3–14). Taipei, Taiwan: IEEE.

Allen, J., Chambers, N., Ferguson, G., Galescu, L., Jung, H., Swift, M., & Taysom, W. (2007). PLOW: A collaborative task learning agent. *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence* (pp. 1514–1519). Vancouver, BC: AAAI Press.

Bailey, T. L., Boden, M., Buske, F. A., Frith, M., Grant, C. E., Clementi, L., Ren, J., Li, W. W., & Noble, W. S. (2009). MEME SUITE: Tools for motif discovery and searching. *Nucleic Acids Research*, *37*, W202–W208.

Blythe, J. (2005). Task learning by instruction in Tailor. *Proceedings of the Tenth International Conference on Intelligent User Interfaces* (pp. 191–198). San Diego, CA: ACM.

Chand, C., Thakkar, A., & Ganatra, A. (2013). Sequential pattern mining: Survey and current research challenges. *International Journal of Soft computing and Engineering, 2*, 185–193.

Chou, M. F., & Schwartz, D. (2011). Biological sequence motif discovery using motif-x. *Current Protocols in Bioinformatics*, *35*, 13.15.1–13.15.24

Cook, J. E., & Wolf, A. L. (1998). Discovering models of software processes from event-based data. *ACM Transactions on Software Engineering and Methodology*, *7*, 215–249.

Eker, S., Lee, T. J., & Gervasio, M. (2009). Iteration learning by demonstration. *Papers from the AAAI 2009 Spring Symposium on Agents that Learn from Human Teachers* (pp. 40–47). Stanford, CA: AAAI Press.

Etzioni, O. (1993). Acquiring search-control knowledge via static analysis. *Artificial Intelligence*, *62*, 255–301.

Fikes, R. E., Hart, P. E., & Nilsson, H. J. (1972). Learning and executing generalized robot plans. *Artificial Intelligence*, *3*, 251–288.

Fikes, R. E., & Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, *2*, 189–208.

Fournier-Viger, P., Gomariz, A., Campos, M., & Thomas, R. (2014). Fast vertical mining of sequential patterns using co-occurrence information. *Proceedings of the Eighteenth Pacific-Asia Conference on Knowledge Discovery and Data Mining* (pp. 40–52). Tainan, Taiwan: Springer.

Fournier-Viger, P., Lin, J. C.-W., Gomariz, A., Gueniche, T., Soltani, A., Deng, Z., & Lam, H. T. (2016). The SPMF OpenSource data mining library version 2. *Proceedings of the Nineteenth European Conference on Principles of Data Mining and Knowledge Discovery* (pp. 36–40). Riva del Garda, Italy: Springer.

Fournier-Viger, P., Lin, J. C.-W., Kiran, R. U., Koh, Y. S., & Thomas, R. (2017). A survey of sequential pattern mining. *Data Science and Pattern Recognition*, *1*, 54–77.

Garvey, T., Gervasio, M., Lee, T., Myers, K., Angiolillo, C., Gaston, M., Knittel, J., & Kolojejchick, J. (2009). Learning by demonstration to support military planning and decision making. *Proceedings of the Twenty-First International Conference on Innovative Applications of Artificial Intelligence* (pp. 1597–1604). Pasadena, CA: AAAI Press.

Gervasio, M., & Lee, T. J. (2013). Discovering action idioms. *Proceedings of the 2013 IEEE Symposium on Visual Languages and Human-Centric Computing* (pp. 11–14). San Jose, CA: IEEE.

Gervasio, M. T., & Murdock, J. L. (2009). What were you thinking? Filling in missing dataflow through inference in learning from demonstration. *Proceedings of the Fourteenth International Conference on Intelligent User Interfaces* (pp. 157–166). Sanibel Island, FL: ACM.

Ghallab, M., Nau, D., & Traverso, P. (2004). *Automated planning: Theory and practice*. San Francisco, CA: Morgan Kaufmann.

Gratch, J., & DeJong, G. (1991). A hybrid approach to guaranteed effective control strategies. *Proceedings of the Eighth International Conference on Machine Learning* (pp. 509–513). Evanston, IL: Morgan Kaufmann.

Korf, R. E. (1985). Macro-operators: A weak method for learning. *Artificial Intelligence*, *26*, 35–77.

Kovacs, D. L. (2011). *BNF definition of PDDL 3.1*. Unpublished manuscript from the IPC-2011 website. https://helios.hud.ac.uk/scommv/IPC-14/repository/ kovacs-pddl-3.1-2011.pdf

Iba, G. A. (1989). A heuristic approach to the discovery of macro-operators. *Machine Learning, 1*, 285–317.

Laird, J. E., Rosenbloom, P. S., & Newell, A. (1986). Chunking in Soar: The anatomy of a general learning mechanism. *Machine Learning*, *1*, 11–46.

Li, N., Stracuzzi, D. J., Langley, P., & Nejati, N. (2009). Learning hierarchical skills from problem solutions using means-ends analysis. *Proceedings of the Thirty-First Annual Meeting of the Cognitive Science Society* (pp. 1858–1863). Amsterdam: Cognitive Science Society.

McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., & Wilkins, D. (1998). *PDDL–The Planning Domain Definition Language*. CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control.

Minton, S. (1985). Selectively generalizing plans for problem-solving. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*. Los Angeles, CA: AAAI.

Minton, S., Carbonell, J. G., Knoblock, C. A., Kuokka, D. R., Etzioni, O., and Gil, Y. (1989). Explanation-based learning: A problem solving perspective. *Artificial Intelligence*, *40*, 63–118.

Mooney, R. J. (1988). Generalizing the order of operators in macro-operators. *Proceedings of the Fifth International Conference on Machine Learning* (pp. 596–599). Ann Arbor, MI: Morgan Kaufmann.

Muñoz-Gama, J. (2014). *Large bank transaction process*. Universitat Politècnica de Catalunya Data Set. https://doi.org/10.4121/uuid:c1d1fdbb-72df-470d-9315-d6f97e1d7c7c

Myers, K., Kolojejchick, J., Angiolillo, C., Cummings, T., Garvey, T., Gervasio, M., Haines, W., Jones, C., Knittel, J., Morley, D., Ommert, W., & Potter, S. (2011). Learning by demonstration technology for military planning and decision making: A deployment story. *Proceedings of the Twenty-Third International Conference on Innovative Applications of Artificial Intelligence* (pp. 1597–1604). San Francisco, CA: AAAI Press.

Negrevergne, B., & Guns, T. (2015). Constraint-based sequence mining using constraint programming. *Proceedings of the Twelfth International Conference on Integration of Artificial Intelligence and Operations Research in Constraint Programming for Combinatorial* (pp. 288–305). Barcelona, Spain: Springer.

Pei, J., Han, J., Mortazavi-Asl, B., Wang, J., Pinto, H., Chen, Q., Dayal, U., & Hsu, M. C. (2004). Mining sequential patterns by pattern-growth: The PrefixSpan approach. *IEEE Transactions on Knowledge and Data Engineering*, *16*, 1424–1440.

Pei, J., Han, J., & Wang, W. (2002). Mining sequential patterns with constraints in large databases. *Proceedings of the Eleventh International Conference on Information and Knowledge Management* (pp. 18–25). McLean, VA: ACM.

Peintner, B., Dinger, J., Rodriguez, A., & Myers, K. (2009). Task Assistant: Personalized task management for military environments. *Proceedings of the Twenty-First International Conference on Innovative Applications of Artificial Intelligence*. Pasadena, CA: AAAI Press.

Rojas, E., Muñoz-Gama, J., Sepúlveda, M., & Capurro, D. (2016). Process mining in healthcare: A literature review. *Journal of Biomedical Informatics*, *61*, 224–236.

Seegebarth, B., Schattenberg, B., & Biundo, S. (2012). Making hybrid plans more clear to human users—A formal approach for generating sound explanations. *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling* (pp. 225–233). São Paulo, Brazil: AAAI Press.

Shell, P. & Carbonell, J. G. (1989). Towards a general framework for composing disjunctive and iterative macro-operators. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 596–602). Detroit, MI: AAAI.

Srikant, R. & Agrawal, R. (1996). Mining sequential patterns: Generalization and performance improvements. *Proceedings of the International Conference on Extending Database Technology*, (pp. 1–17). Berlin: Springer.

van der Aalst, W. (2012). Process mining: Overview and opportunities. *ACM Transactions on Management Information Systems*, *3*, 1–17.

van der Aalst, W. M. P., Reijers, H. A., Weijters, A. J. M. M. van Dongen, B. F., Alves De Medeiros, A. K., Song, M., & Verbeek, H. M. W. (2007). Business process mining: An industrial application. *Information Systems*, *32*, 713–732.

van Lent, M., & Laird, J. E. (2001). Learning procedural knowledge through observation. *Proceedings of the First International Conference on Knowledge Capture* (pp. 179–186). Victoria, BC: ACM.

Zaki, M. J. (2001). SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning*, *42*, 31–60.