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 cognitive perspective. In contrast, humans are able to form models of procedures from small numbers of observations, even without explicit instruction. We posit that humans are able to do so because of background knowledge about actions and procedures, which lets them effectively filter out meaningless sequential patterns. The action models foundational to artificial intelligence (AI) planning is one way to provide semantics to 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 the value of the filters in greatly reducing the number of sequential patterns discovered without sacrificing the number of correct patterns found, even with small, noisy event logs.

1. Introduction

Humans are quite 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 their new 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 are able to recognize meaningful patterns in observed behavior even without explicit demonstration or instruction. And we can do so without requiring large numbers of examples. We are able to identify what is relevant and what is not so that even with just a few examples of some unknown procedure being executed, we are able to learn the underlying process.

A popular computational technique for discovering sequential patterns from multiple examples is sequence mining. In the data mining community, sequence mining has been applied to discover patterns in consumer behavior (Agrawal & Srikant, 1995) to be used, for example, in marketing campaigns to recommend products a consumer is likely to be interested in. In bioinformatics, sequence mining is used to discover motifs—gene and protein sequences that have distinct functions (Abouelhoda & Ghanem, 2010). 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 the work on process mining, which attempts to discover process knowledge from

event logs, primarily for the purpose of analysis (van der Aalst et al., 2012). All these applications of sequence mining operate on large volumes of data and rely on frequency as the main indicator of a pattern. They are interested in finding frequent patterns that signify general trends. In many cases, the 'correctness' of a pattern is immaterial—it doesn't really matter if a purchase pattern is not particularly meaningful or a discovered biological motif or process workflow is not quite right; they are often generated as possibilities or starting points for a human expert to analyze or refine.

In contrast, 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 also rich in structure. They have parameters (often typed), preconditions, and effects; they take inputs and generate outputs; they are organized in hierarchies; and so on. 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.

AI planning relies on action models to construct plans for achieving particular goal conditions from a given set of initial conditions. These formal specifications of actions enable the reasoning required to determine which actions can be executed in a given state and the effects they will have, enabling an automated planner to generate a sequence of actions to achieve a goal (Ghallab et al., 2004). In the work described in this paper, we set out to investigate whether we could augment sequence mining techniques with filtering heuristics derived from planning knowledge to help identify promising candidate sequences. Using the action models, we devised two sets of heuristics—a set of action filters to eliminate noise from discovered patterns, and a set of candidate filters to eliminate undesirable discovered patterns— and we conducted experiments to evaluate the ability of these filters to improve precision without sacrificing recall.

We begin with a discussion of related work in procedural learning and sequence mining. We then present the sequential pattern mining problem and our basic approach to discovering frequent parameterized action sequences. We discuss experiments on a bank transaction dataset and present results illustrating the problem with using simple frequency-based techniques to find candidate procedures—a problem exacerbated by the presence of noise. We then introduce action models and present the motivation behind the filters we designed to eliminate undesirable candidates. We discuss experiments with the filters on both the noise-free and noisy datasets, showing the effectiveness of the filters in increasing precision, with minimal decrease in recall. The work is a first attempt at using planning knowledge to improve the results of sequence mining for procedure learning and we conclude with a discussion of future work.

2. 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 have been explored, including learning from problem-solving (Laird et al., 1986), learning from observation (van Lent & Rosenbloom, 2001), learning from instruction (Blythe, 2005), multi-modal learning (Allen et al., 2007), learning from demonstration (Gervasio & Murdock, 2009), and learning from solution traces (Li et al., 2009). The work in this paper addresses the same problem, but differs in the use of event logs as the source of data from which to learn procedures. The event logs capture the actions executed by one or more agents rather than actions of the learning agent itself or the actions of an expert and they are not guaranteed to be complete, clean (i.e., noise-free), or optimal. As such, we use as our starting point the work in sequential pattern mining.

In the AI planning community, the concept of macro-operators was conceived as a means for compiling the search involved to determine a sequence of actions to achieve a goal from a given initial state, simplifying planning in similar future situations (Fikes et al., 1972). Later work has explored macro learning for iterative and recursive plans (Shavlik, 1990), partially ordered plans (Botea et al., 2005), hierarchical task networks (Hogg et al., 2014), and arbitrary planners and domains (Newton et al., 2007). The procedures discovered in our work are driven by a similar motivation of efficiency but are learned from logs of user actions that can interleave tasks and may be suboptimal or noisy, in contrast to the correct-by-construction sample plans for a single goal that underpin macro learning.

The idea of finding sequential patterns in data has been explored in a variety of ways in a number of fields. In the data mining community, sequential pattern mining was introduced by Agrawal and Srikant (1995) in their seminal work on algorithms for market basket analysis. Frequent sequential patterns discovered in databases of customer transactions can be used to drive various decisions about marketing activities. A multitude of sequential pattern mining algorithms have been developed since then (e.g., Fournier-Viger et al., 2014; Pei et al., 2004; Srikant & Agrawal, 1996; Zaki, 2001), differing in how they search the space of patterns, how they represent the database, how they generate next candidates, and how they determine the support (frequency of occurrence) for a pattern (Fournier-Viger et al., 2017). Because the applications driving the work in the data mining community involve very large databases, the research in this area has focused primarily on time and space efficiency.

Sequence mining in bioinformatics (Abouelhoda & Ghanem, 2010) similarly involves very large databases and thus emphasizes highly efficient algorithms. However, while data mining problems often involve finding patterns in large numbers of relatively short sequences, sometimes involving a very large number of items, biological sequence mining involves finding patterns in very long sequences from relatively small alphabets. Furthermore, the main driver for biological sequence mining is finding repeated strings that correspond to some significant biological structure or function—i.e., motifs (e.g., Bailey et al., 2009; Chou & Schwartz, 2011).

Process mining is concerned with the analysis of processes based on event logs (van der Aalst, 2012). Growing out of the work on business process modeling, much of the work in this area is concerned with the discovery of processes for the purposes of process modeling, conformance checking, or workflow enhancement. Because real-life processes can often be quite complex, process mining is designed to discover more complex control structures such as loops and conditionals—for example, a popular representation of a learned model is a Petri net. Process mining has been applied to a variety of domains, assorted healthcare applications (Rojas et al., 2016), software development (Cook & Wolf, 1998), public works infrastructure (van der Aalst et al., 2007), and various other business processes.

Although process mining is not limited to analyzing simple events, its primary focus is on discovering control flow (van der Aalst, 2012). Thus, the consideration of other attributes of the events such as actors, timestamps, and resources is typically done outside of the mining process itself. Constraint-based mining (Negrevergne & Guns, 2015; Pei et al., 2004; Pei & Wang, 2002) provides a possible avenue for biasing the search for frequent patterns by requiring that they satisfy user-specified constraints. However, the constraints addressed have typically been limited in scope, focused on syntactic features of sequential patterns. Negrevergne & Guns (2015) categorize constraints into four types: constraints on patterns (e.g., minimum size), constraints on the cover set (e.g., minimum frequency), constraints on the inclusion relation (e.g., maximum gap), and preferences over the candidate patterns (e.g., maximal patterns). Most work on sequence mining

focuses on developing specialized algorithms for a select subset of these constraints. We can leverage this work but require a new approach to utilize the semantic constraints that govern the action sequences of task-oriented procedures.

3. 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 a variety of real-life problems, including 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 pattern mining algorithms specify a *minimum support* parameter that denotes the minimum frequency of occurrence for a pattern to be considered a viable candidate. And, as described in the previous section, while some algorithms are able to accommodate additional constraints, these are still limited to relatively shallow properties rather than any knowledge about what makes a good pattern.

The sequence mining algorithms today are robust and highly efficient, but they have two main limitations when applied to procedural learning. First, sequence mining algorithms operate over sequences of atoms—e.g., DNA sequences, items purchased by an online customer, URL clickstreams, event logs. In contrast, procedure mining should be designed for input sequences composed of *parameterized actions* and should be able to discover relationships between those parameters to achieve parameter generalization. Second, there are no semantics associated with these atoms—in fact, they are often simply converted into integers for compactness. Because the 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, they manipulate data, etc.

The work described in the remainder of this paper addresses the second limitation. Here, we first review how we address the first limitation using the technique described in (Gervasio & Lee, 2013). 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 action with one input parameter (the

Table 1. Sequential pattern mining is limited to finding patterns over actions (without 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 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 ApproveID21	*Sequence 2, Support 1*: Get(Name1,Id2) Approve(Id3)
Get(Jill,ID37) Approve(ID37)		*Sequence 3, Support 1*: GetJillID37 ApproveID37	

name of the person whose ID to retrieve) and one output parameter (the person's ID). *Approve* is an approval action taking 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 is a different procedure involving the retrieval of one person's ID and the approval of another.

A straight application of sequence mining would look only at the action names (*Get* and *Approve*), resulting in discovering 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 results in the opposite problem of under-generalization, with each input sequence being recognized as a separate pattern (Table 1, *Result 2*). The desired result is one that distinguishes between the patterns with recognition of the relationships between the action arguments as well (Table 1, *Desired Result*).

As in (Gervasio & Lee, 2013), to achieve the desired parameter generalization, we first apply sequence mining on the actions only (Table 1, *Result 1*). We then apply 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; and also 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 our basic procedure mining algorithm, which we use in the baseline experiment described next.

4. Baseline Experiment

4.1 Data

To evaluate our approach, we repurposed transactional log datasets collected by the IEEE Task Force on Process Mining and made available through the 4TU.Centre for Research Data (2016). Specifically, we used the Large Bank Transaction Process dataset (Muñoz-Gama, 2014), a collection of synthetic event logs generated from a large model of bank transactions. The IEEE Task Force collection includes both real-life and synthetic logs. We chose to work with the synthetic logs because they included the Petri net model from which the logs were generated, enabling us to develop "ground truth" data (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; this is the portion we focused on in our experiments. From here, we extracted 28 target (ground truth) procedures, corresponding to the paths described in Figure 3.

Like most transactional logs, each entry in the bank transaction dataset consists simply of a log identifier and an event type. To transform the data into a form more akin to the action logs we would obtain from a system that logs parameterized actions, we modeled each action (square) in Figure 3 in terms of its inputs and outputs. We then translated each event in the dataset to its equivalent action, generating the action log we used in our experiments. Figure 4 illustrates this transformation. The dataset provided a few different synthetically generated logs. For our experiments, we used the one comprising 2000 noise-free logs of observed transaction sequences

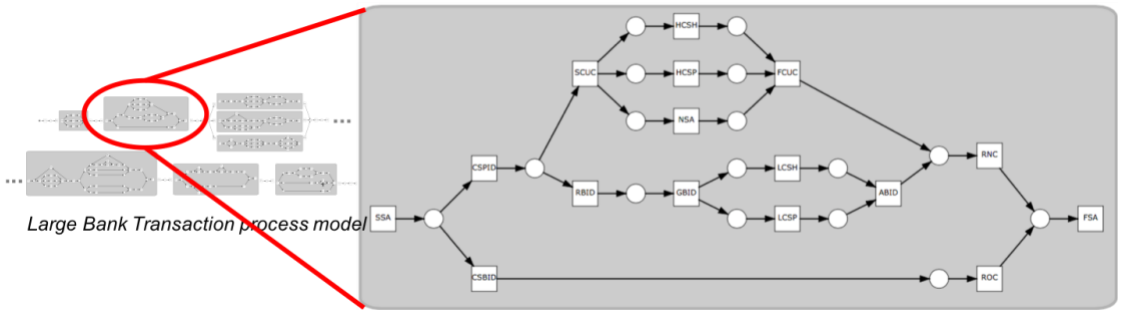

Figure 1. Large Bank Transaction Petri Net Model. Callout shows the Sender (Customer) Authentication subprocess used in the experiments discussed in this paper. (The complete model on the left is shown only to provide context; its details are irrelevant.)

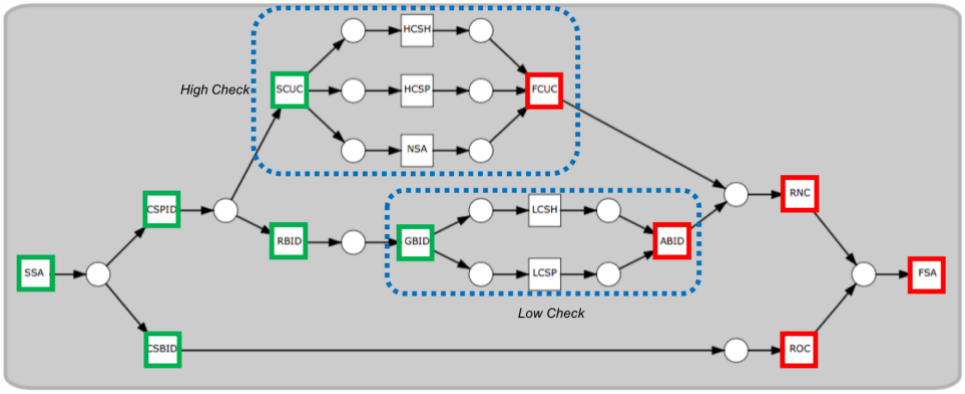

Figure 2. The target (ground truth) procedures in the *Sender Authentication* subprocess consist of all possible paths between a green square and a red square. The diagram is a Petri net so all the branches emanating from a square must be traversed in any order This leads to six different sequences for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

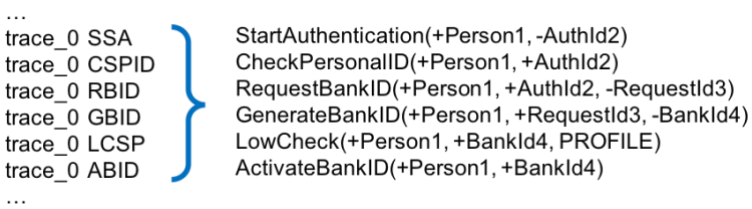

Figure 3. Example conversion from event log to action log. Each atomic event is translated into its equivalent parameterized action.

(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 from this dataset. We refer to this dataset as the noise-free dataset.

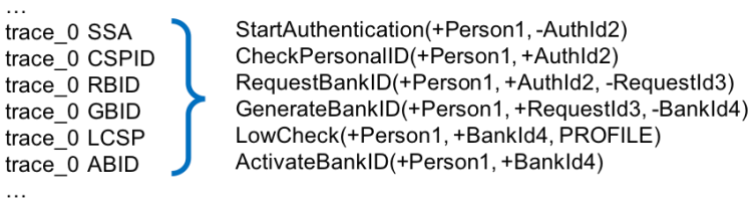

Figure 4. Example conversion from event log to action log. Each atomic event is translated into its equivalent parameterized action.

To create noisy versions of this dataset, we injected two types of noise: redundant actions and extraneous actions.[1] We injected redundant actions by going through each action in the log, and making a decision on whether to repeat that action with probability p. Similarly, we inserted extraneous actions by going through each action and deciding to insert a randomly selected action with probability *p*. Using $p = 0.1$, we created three noisy datasets: one with *redundant* actions only, one with *extraneous* actions only, and one with *both* redundant and extraneous actions.[2]

4.2 Experimental Setup and Evaluation Metrics

Given the ground truth sequences, we were interested in how well the candidate sequences discovered by sequence mining would match ground truth. Thus, we measured *precision* and *recall*. There is an argument to be made that precision is more important than recall when learning procedures from observation, so we report two *F* measures: the F_1 score, which weights precision and recall evenly; and the $F_{0.5}$ score, which weights precision twice as much as recall.

We expect 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 many spurious procedures to be found and thus precision to be low. This subsumption property is what led much work in frequent pattern mining to focus on finding only maximal sequences (i.e., frequent sequences which are not contained in a longer frequent sequence) or closed sequences (i.e., maximal sequences which are not contained in a longer maximal sequence with exactly the same support). For procedure mining, however, we intentionally do not want to limit ourselves to closed or maximal sequences because there are likely to be useful subsequences as well, as seen in the ground truth procedures.

To establish a baseline, we ran the CM-SPAM[3] algorithm (Fournier-Viger et al., 2014) over the transformed action log with a minimum support of 5. Table 2 summarizes the results for the *Noise-*

1 We also considered missing actions but decided this did not make sense in the procedural automation setting because action traces that were missing actions would not have accomplished the desired effects and are thus not really examples of the target procedure.

2 We tried both smaller and larger values of *p*. These resulted in the expected decrease and increase, respectively, in the number of candidates generated but the overall results on the effects of using filters did not change.

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

Table 2. 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 the number of *correct* candidates found.

Dataset	#Cand	True	Recall	Prec	F_1	$F_{0.5}$
Noise-free	229	28	1.000	0.1223	0.2179	0.1483
Redundant	289	28	1.000	0.0720	0.1343	0.0884
Extraneous	252	25	0.8929	0.0992	0.1786	0.1206
Both	372	19	0.6786	0.0511	0.0950	0.0627

free case and the three noisy datasets (*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 datasets, with more incorrect candidates found in all three and fewer correct candidates found in the *Extraneous* and *Both* noisy datasets as well.

5. 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. They may require (non-observable) intervening actions, include systematic noise, include non-automatable actions, and so on. Extraneous actions which serve no purpose cannot be filtered out, nor can nonsensical action sequences such as those that start with actions with preconditions that have not been established. The action models used in AI planning are designed precisely to enable reasoning about such dependencies between actions. With this in mind, we set out to see how we could use action models to filter out the false candidates from the large number of candidates generated by sequential pattern mining algorithms.

5.1 Action Model

We use a hybrid action model that combines standard STRIPS-style semantics capturing preconditions and effects with a dataflow-oriented representation of actions in terms of their inputs and outputs.

5.1.1 Planning Domain Definition Language.

The Planning Domain Definition Language (PDDL) (Kovacs, 2011; McDermott et al., 1998), originated from 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 and a set of preconditions and effects which define, respectively, what is required for and what results from the execution of the action. For example, Figure 5 (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 authentication

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

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

Figure 5. PDDL specification of *FinishAuthentication* action (left) and dataflow-based specification of *StartAuthentication* action (right).

be in process and that the customer already have been registered, and results in the sender being authenticated.

5.1.2 Task Learning Actions

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 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 developed a representation where 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, Figure 5 (right) shows the action definition for *StartAuthentication*: Given as input a sender (name), it outputs a new authentication ID.

5.1.3 Hybrid Action Model

We extended our existing task learning action model with preconditions and effects as in PDDL to encode the planning knowledge for a domain. We note that prior 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. However, action or transactional logs typically lack such state information. And yet humans are able to look at such action sequences and infer the intervening states based on their knowledge of the actions. We are also able to identify related actions and to ignore irrelevant ones. This was the primary motivation behind our investigation into the use of action-model-based heuristics to filter the candidate patterns discovered through sequential pattern mining.

5.2 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

Table 3. Candidate filters.

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

based on common-sense knowledge about the nature of actions and procedures in this domain. These filters, summarized in Table 3, are meant primarily to serve as examples and to evaluate the idea of knowledge-based filtering; they are not meant to be complete or definitive.

The *Precondition* filter discards any candidate containing an action whose preconditions are not satisfied. This corresponds to the fact that a procedure cannot be executed to completion if any precondition of any of its actions is not satisfied. Since event logs do not contain state information, we rely on information about acceptable initial state conditions instead. To signify these conditions that 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)* (i.e., 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, requiring that every action that starts a process has a corresponding action that ends it.

The final candidate filter, *Branch*, discards candidates that do not begin with an action recognized to be one of multiple options. This is a filter that may not apply to procedures in many other domains but in the banking transaction domain, we know that there are often steps that can be executed in any order—i.e., each of the steps can effectively start a procedure consisting of all the step in some order. 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 one of these branching actions is likely to correspond to a good candidate.

In addition to the candidate filters in Table 3, we also devised action filters for discarding unnecessary actions in candidates; these filters are summarized in Table 4. 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

Table 4. Action filters.

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

Table 5. Combination Filters

Filter	Description
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

rationale being that repeating the exact same action serves no purpose. These action filters are designed to eliminate 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 utilizes the maximum gap constraint in pattern mining algorithms such as CM-SPAM, which is designed to accommodate noise by allowing gaps to a certain length within a sequence. For the experiments described below, we allowed up to one intervening action between any two elements of a candidate sequence, which we implemented by setting the built-in CM-SPAM maximum gap parameter accordingly.

Based on the recognition that the *Precondition* and *Contribution* filters check for *necessary* conditions while the rest checked for *desirable* ones, we also devised two combinations of filters that require passing the *Contribution* and *Precondition* filters and at least one of the other candidate filters. These are described in (Table 5).

6. Experimental Results

6.1 Noise-Free Data

Our main hypothesis was that the filters would remove bad candidates from consideration, thereby improving precision. However, we were also interested in whether recall would suffer and by how much. Table 6 summarizes the results with the use of the different filters, alone and in combination, applied over the frequent sequences found by CM-SPAM on the noise-free dataset. The individual candidate filters never degrade precision and *F* scores and in most cases, improve them, although recall is sometimes affected. Among the individual filters, the *Branch* filter results in the highest precision, the *Complete* filter in the highest F_1 score, and both filters in the highest $F_{0.5}$ score. However, the *Branch* filter has markedly lower recall. The best performance overall, however, is achieved by the *Combo1* filter, which records the highest precision and *F*-scores, along with almost perfect recall.

Table 6. Results of filtering on noise-free dataset.

Filter	#Cands	True	Recall	Prec	F_1	$F_{0.5}$
Precondition	229	28	1.000	0.1223	0.2179	0.1483
Start	94	17	0.6071	0.1809	0.2787	0.2104
Finalize	87	15	0.5357	0.1724	0.2609	0.1994
Complete	156	28	1.000	0.1795	0.3043	0.2147
Branch	92	17	0.6071	0.1848	0.2833	0.2147
Contribution	215	28	1.000	0.1302	0.2305	0.1576
Duplicate	229	28	1.000	0.1223	0.2179	0.1483
Gap	1020	28	1.000	0.0275	0.0534	0.0341
Combo1	128	26	0.9286	0.2031	0.3333	0.2407
Combo2	169	28	1.0000	0.1657	0.2843	0.1989

Table 7. Results of filtering on noisy dataset with *Redundant* actions.

Filter	#Cands	True	Recall	Prec	F_1	$F_{0.5}$
Precondition	251	28	1.0000	0.1116	0.2007	0.1357
Start	156	17	0.6071	0.1090	0.1848	0.1304
Finalize	134	15	0.5357	0.1119	0.1852	0.1329
Complete	272	28	1.0000	0.1029	0.1867	0.1254
Branch	149	17	0.6071	0.1141	0.1921	0.0961
Contribution	357	28	1.000	0.0784	0.1455	0.1362
Duplicate	229	28	1.000	0.1223	0.2179	0.1483
Gap	2201	28	1.0000	0.0127	0.0251	0.0158
Combo1	128	26	0.9286	0.2031	0.3333	0.2407
Combo2	169	28	1.0000	0.1657	0.2843	0.1989

The *Contribution*, *Duplicate*, and *Gap* action filters were designed primarily to address noise and so were not expected to help much in the noise-free case. Because the noise-free dataset contained no repeated actions, the *Duplicate* filter also 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 affecting performance. The *Contribution* filter has a mild positive effect on performance, removing a small number of false positives for a slight improvement in precision.

6.2 Noisy Data

Table 7 summarizes the results of applying the different filters on the noisy dataset with *Redundant* actions only. With the exception of the *Gap* filter, every filter improved precision and *F*-scores. Not surprisingly, the *Duplicate* filter, which removes repeated actions (i.e., exactly the injected

Table 8. Results of filtering on noisy dataset with *Extraneous* actions.

Filter	#Cands	True	Recall	Prec	F_1	$F_{0.5}$
Precondition	232	25	0.8929	0.1078	0.1923	0.1308
Start	99	15	0.5357	0.1515	0.2362	0.1769
Finalize	89	13	0.4643	0.1461	0.2222	0.1693
Complete	159	25	0.8929	0.1572	0.2674	0.1882
Branch	98	15	0.5357	0.1531	0.2381	0.1786
Contribution	208	25	0.8929	0.1202	0.2119	0.1454
Duplicate	251	25	0.8929	0.0996	0.1792	0.1211
Gap	1266	28	1.0000	0.0221	0.0433	0.0274
Combo1	116	26	0.8214	0.1983	0.3194	0.2338
Combo2	159	28	0.8929	0.1572	0.2674	0.1882

noise), resulted in the best precision on the *Redundant* dataset, matching the performance of the baseline in the noise-free case. And applying the *Duplicate* filter in combination with any of the other filters 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, significantly increased the number of candidates, resulting in greatly degraded precision and *F*-scores. 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 finds many other extra candidates. We had expected the *Gap* filter to have some positive effect on small datasets having patterns that have frequencies of occurrence close to the minimum, but the extremely high number of other patterns found makes it unlikely that it will be useful.

Table 8 shows the results for the different filters on the noisy dataset with *Extraneous* actions only, and Table 9 shows the results for the noisy dataset with *Both* redundant and extraneous actions. The results are similar to those for noise due to *Redundant* actions only. Again, with the exception of the *Gap* filter, all the filters have a positive effect on performance, with the combination filters resulting in the greatest improvement.

6.3 Discussion

The experimental results confirm that standard sequential pattern mining techniques discover a large number of irrelevant action sequences and that action-model-based filters help eliminate many of them. Every candidate filter and all but the *Gap* action filter was able to eliminate irrelevant sequences and most were able to do so without overly sacrificing recall. Baseline precision is just over 12% in the noise-free case and ranges from 5–10% in the noisy datasets. With the filters, this reaches up to 20% in the noise-free case and up to 15–20% for the noisy datasets. Similarly, the baseline $F_{0.5}$ score is under 15% in the noise-free case and ranges from around 9–12% in the noisy datasets. With the filters, this reaches up to 24% in the noise-free case and up to 18–24% for the

Table 9. Results of filtering on noisy dataset with both *Redundant* and *Extraneous* actions.

Filter	#Cands	True	Recall	Prec	F_1	$F_{0.5}$
Precondition	218	19	0.6786	0.0872	0.1545	0.1056
Start	135	11	0.3929	0.0815	0.1350	0.0968
Finalize	131	9	0.3214	0.0687	0.1132	0.0815
Complete	252	19	0.6786	0.0754	0.1357	0.0917
Branch	134	12	0.4286	0.0896	0.1481	0.1064
Contribution	296	19	0.6786	0.0642	0.0950	0.0627
Duplicate	268	26	0.9286	0.0970	0.1757	0.1182
Gap	1339	28	1.0000	0.0209	0.0410	0.0260
Combo1	108	17	0.6071	0.1574	0.2599	0.1847
Combo2	149	19	0.6786	0.1275	0.2147	0.1522

noisy datasets. Although this marks a 100–200% improvement, it remains to be seen whether a 1 in 5 hit rate for discovered procedures intended for automation is sufficient. Nevertheless, these results show that knowledge-based heuristics for filtering candidate patterns and pattern actions can be an effective addition to standard sequential pattern mining to guide discovery toward meaningful action sequences that are more likely to correspond to procedures.

7. Conclusions and Future Work

In this paper, we proposed a hybrid approach for 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. The approach leverages efficient sequence mining techniques to find frequent action sequences from logs of user actions to serve as candidate action sequences for automation. Using action models that provide a semantic representation of actions in terms of their preconditions and effects and inputs and outputs, we devised filtering heuristics to help identify good candidate sequences for automation. 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. We conducted several experiments to evaluate the usefulness of the filters on both noise-free and noisy datasets and the experimental results show that the action-model-based filtering heuristics successfully eliminate a large number of irrelevant sequences discovered by standard sequential pattern mining.

Our approach does introduce an additional cost to deployment above and beyond that of purely statistical methods in that it requires the formulation of the action model 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 we presented in this paper may not apply to all domains all the time. Some, like the *Branch* filter, take advantage of characteristics specific to the banking transaction domain. On the other hand, we are likely to be able to leverage characteristics specific to any domain by designing new filters, corresponding to the knowledge one would have about processes in that domain. 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 all include retrieving resolution options based on a standard operating manual or prior similar incidents. And they may all conclude with either a resolution or an escalation.

It is also likely that we will have to learn from even smaller numbers of examples. This will need filters like the *Gap* filter to enable finding sequences for which there may not be enough support otherwise. For example, an abstraction filter that recognizes a set of actions to be variants of each other because they achieve the same cumulative effects would allow sequences that differed only in the variant used to be grouped together.

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 (i.e., its accumulated effects). We can potentially use this information to generate explanations to a user to accompany suggestions for task automation, drawing on explanatory techniques such as those described in (Seegebarth et al., 2012).

Our work is motivated by the ultimate objective of enabling process automation for real-world usage. Our focus to date has been on a collaborative task management tool that we developed previously called Task Assistant (Peintner et al., 2009). Task Assistant supports distributed human teams in collectively executing complex coordinated processes (e.g., Standard Operating Procedures (SOPs)) 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 to perform frequently performed support tasks, many of which focus on information retrieval to aid human decision making. Our first approach to procedure automation for Task Assistant involved learning from demonstration technology (Myers et al., 2011). While useful, the approach still required significant user effort to recognize the need for automation and to explicitly demonstrate the procedures to be automated. Our engagement with the user community has shown a strong desire for approaches that can automate procedures with minimal human intervention, as would be enabled by the procedure mining technique introduced in this paper.

Acknowledgments

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, Heidelberg: Springer-Verlag,.

Agrawal, R. & Srikant, R. (1995). Mining sequential patterns. *Proceedings of the International Conference on Data Engineering* (ICDE).

Allen, J., Chambers, N., Ferguson, G., Galescu, L., Jung, H., Swift, M., & Taysom, W. (2007). PLOW: A collaborative task learning agent. *Proceedings of the 27th AAAI Conference on Artificial Intelligence* (pp. 1514–1519).

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*(*suppl_2*), W202–W208.

Blythe, J. (2005). Task learning by instruction in Tailor. *Proceedings of the 10th International Cconference on Intelligent User Interfaces* (pp. 191–198).

Botea, A., Müller, M., & Schaeffer, J. (2005). Learning partial-order macros from solutions.. *Proceedings of the 15th International Conference on Automated Planning and Scheduling* (pp. 231–240).

Chand, C., Thakkar, A., & Ganatra, A. (2013). Sequential pattern mining: Survey and current research challenges. *International Journal of Soft computing and Engineering*, *2*(1), 185–193.

Chou, M. F. & Schwartz, D. (2011). Biological sequence motif discovery using motif-x. *Current Protocols in Bioinformatics*, *35*(1).

Cook, J. E. & Wolf, A. L. (1998). Discovering models of software processes from event-based data. *ACM Transactions on Software Engineering and Methodology* (TOSEM), *7*(3), 215–249.

Eker, S., Lee, T. J., & Gervasio, M. (2009). Iteration learning by demonstration. *Papers from the AAAI 2009 Spring Sympsoium on Agents that Learn from Human Teachers*.

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*(3–4), 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 Pacific-Asia Conference on Knowledge Discovery and Data Mining* (pp. 40–52).

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 19th European Conference on Principles of Data Mining and Knowledge Discovery*, 36–40.

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*(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 21st International Conference on Innovative Applications of Artificial Intelligence*.

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).

Gervasio, M. T. & Murdock, J. L. (2009). What were you thinking? Filling in missing dataflow through inference in learning from demonstration. *Proceedings of the 14th International Conference on Intelligent User Interfaces*.

Ghallab, M., Nau, D., & Traverso, P. (2004). *Automated Planning: Theory and Practice*. Elsevier.

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

Hogg, C., Munoz-Avila, H., & Kuter, U. (2014). Learning hierarchical task models from input traces. *Computational Intelligence*, *32*(1), 3-48.

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

Li, N., Stracuzzi, D. J., Langley, P., and Nejati, N. (). Learning hierarchical skills from problem solutions using means-ends analysis. *Proceedings of the Annual Meeting of the Cognitive Science Society*, *31*(31), 1858–1863.

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.

Muñoz-Gama, J. (2014). *Large Bank Transaction Process*. Universitat Politècnica de Catalunya (Barcelonatech). Dataset. 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 23rd International Conference on Innovative Applications of Artificial Intelligence*.

Negrevergne, B. & Guns, T. (2015). Constraint-based sequence mining using constraint programming. *Proceedings of the International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems* (pp. 288–305), Springer.

Newton, M. A. H., Levine, J., Fox, M., & Long, D. (2007). Learning macro-actions for arbitrary planners and domains.. *Proceedings of 17th International Conference on Automated Planning and Scheduling* (pp. 256–263).

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*(11), 1424–1440.

Pei, J., Han, J., & Wang, W. (2002). Mining sequential patterns with constraints in large databases. *Proceedings of 2002 ACM International Conference on Information and Knowledge Management*.

Peintner, B., Dinger, J., Rodriguez, A., & Myers, K. (2009). Task assistant: Personalized task management for military environments. *Proceedings of the 21st International Conference on Innovative Applications of Artificial Intelligence*.

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 22nd International Conference on Automated Planning and Scheduling* (pp. 225–233).

Shavlik, J. W. (1990). Acquiring recursive and iterative concepts with explanation-based learning. *Machine Learning*, *5*(1), 39–70.

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

van der Aalst, W. (2012). Process mining: Overview and opportunities. *ACM Transactions on Management Information Systems (TMIS)*, *3*(2), 7.

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*(5), 713–732.

van Lent, M. & Laird, J. E. (2001). Learning procedural knowledge through observation. *Proceedings of the 1st International Conference on Knowledge Capture* (pp. 179–186).

Zaki, M. J. (2001). SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning*, *42*(1–2), 31–60.