Flexible Heuristics Miner (FHM)

A.J.M.M. Weijters and J.T.S. Ribeiro

School of Industrial Engineering,
Eindhoven University of Technology
P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands.
{a.j.m.m.weijters, j.t.s.ribeiro}@tue.nl

Abstract. One of the aims of process mining is to retrieve a process model from a given event log. However, current techniques have problems when mining processes that contain non-trivial constructs, processes that are low structured and/or dealing with the presence of noise in the event logs. To overcome these problems, a new process representation language (i.e. augmented Causal nets) is presented in combination with an accompanying process mining algorithm. The most significant property of the new representation language is in the way the semantics of splits and joins are represented; by using so-called split/join frequency tables. This results in easy to understand process models even in the case of non-trivial constructs, low structured domains and the presence of noise. The new process representation language and mining technique can also be used for conformance checking; to indicate if all the behavior in the event log is also represented in the process model and if there is extra behavior in the process model not in the event log. This paper explains the new process representation language and how the mining algorithm works. The algorithm is implemented as a plug-in in the ProM framework. An illustrative example with noise and a real life log of a complex and low structured process are used to explicate the presented approach.

Keywords: process mining, work-flow mining, low structured processes, noise

1 Introduction

Modern enterprises are increasingly becoming dependent on the quality of their business processes. This explains why, within organizations, there has been a shift from *data* orientation to *process* orientation. By process we mean the way an organization arranges its work and resources, for instance the order in which tasks are performed and which group of people are allowed to perform specific tasks. A necessary first step to improve business processes is the correct understanding of these processes. *Process mining* [1] aims at the extraction of non-trivial information from running business process data sets (i.e., event logs or transition logs) and can contribute to this understanding. Each entry of an event log represents the sequence of performed tasks – and their details – under a specific business context.

Sometimes, organizations have very explicit process descriptions of the way the work is organized and this description is supported by a process aware information system (PAIS) like, for instance, a work-flow management system (WFM). In

this situation the logs of the PAIS can be used to collect information about the processes as they take place. Analyzing these logs can help to understand the real way of working, because the practical way of working can differ considerably from the prescribed one and even from the intended implementation in the WFM system.

In other situations, there is no, or only a very immature, process description available. However, also in these situations it is often possible to gather information about the processes as they take place. For instance, in many hospitals, information about the different treatments of a patient is registered for reasons like financial administration (date, time, treatment, medical staff). This kind of information, in combination with some process mining techniques, can also be used to get more insight in the health care process [6, 5].

As indicated, event logs are the starting point for process mining techniques. Control-flow mining, conformance checking or performance analysis are possible applications of these techniques. The main focus of the research presented in this paper is on control-flow mining, i.e., the induction of non-trivial process information from running business processes expressed in a process model. However, we will also indicate how the new process representation language and mining technique can also be used for conformance checking.

This paper presents the details of a heuristics-driven control-flow mining algorithm; the so-called "FlexibleHeuristicsMiner" (FHM). It is an updated version of the HeuristicsMiner (HM) [10] as implemented in ProM framework [3]. From practical experiences with the HM during different process mining projects, we learned that not all advantages of the process representation language as used in the HM and the genetic mining approach [7] are completely exploited. In this new version, the FHM, we try to take all the advantages of the underlying basic ideas. The result is an adapted process representation language (i.e., Causal nets (C-nets) and augmented Causal nets (augmented-C-nets)) and an accompanying mining algorithm (FHM). The FHM is implemented in a new version of ProM (version 6.0).

A lot of work in this sub-domain is already done. See [1, 9] for an overview. Most of early solutions try to model *all* the recorded behavior in the event log by using a formal process modeling language (e.g., the Petri net formalism). However these kinds of approach run in problems in low-structured domains such as the ones that can be found in health care applications. The resulting models may easily become unreadable if the model contains a high number of tasks and complex relationships. As an illustration Fig. 1 shows a typical control-flow mining result on an event log of a low-structured domain. The event log contains 2259 cases, 34187 events, and 255 different event classes. The average number of different event classes per case is 14, but some of the cases contain 67 different classes. The term *spaghetti* model used for this kind of results does not need any explanation. On the other hand, simple models like EPC are too vague to provide enough insight in important details of processes. Depending of the mining goals, the challenge is to find good balance between overall structures and details.

Strongly related model representation languages are proposed in [8, 2] as a universal and robust language which allows for accommondating different model semantics, replay semantics, and fitness semantics. However, in [8, 2] they assume that a process model is already available. This discovering is beyond the scope of their papers and is exactly the goal of the FHM as presented in this paper. The most significant difference between the process representations of [8, 2] and the representation as used in this paper is the use of spit/join frequency information in the so-called augmented-C-net. Other relevant work in this domain is done in [4]. However as mentioned in [4] "one of the shortcomings of the presented approach is that it often generates results for which the user cannot understand how they came to be" [page 334]. An important motivation for the approach presented in this paper is the development of a flexible control-flow mining algorithm that performs well in practical situations and with results that are easy to understand.

The remainder of this paper is organized as follows. In Section 2 we first present a process model in the well-known Petri net formalism, that will be used as a running example. In Section 3 we define the new process representation languages (i.e., C-nets). As an illustration the running example is translated into the updated process representation language, the C-net. In Section 4 the details of the different mining steps of FHM are presented: (i) the building of the Dependency Graph (DG), (ii) the extension of the DG up to an augmented-C-net, and (iii) the possible extension of the process model with long-distance dependencies. In Section 5 we illustrate the behavior of the FHM in the situation with noise (5.1), in low structured domains (5.2), and finally we have a closer look at the use of this approach for simple conformance checking (5.3). In the final section (Section 6) we present our conclusion and future work.

2 Running Example

The process model as depicted in Fig. 2 is used as running example to illustrate the mining process of the FHM. This model is also used for generating an artificial

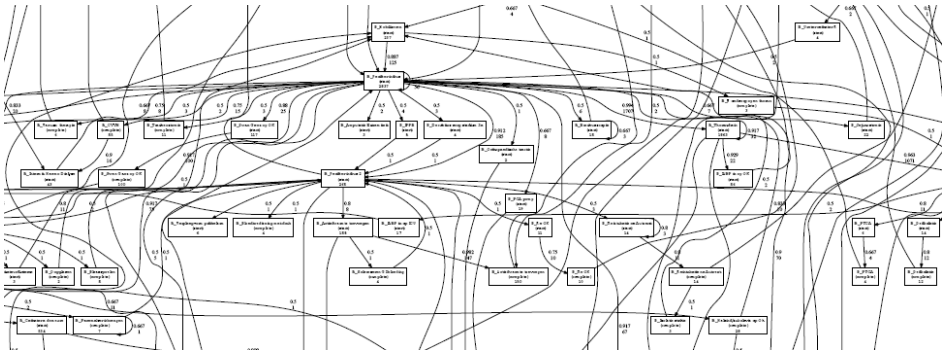

Fig. 1. A typical control-flow mining result on an event log of a low-structured domain.

event log. However, during the generation of the event log, the *hidden tasks* D1, D2, and D3 are not registered. Hidden tasks are a standard solution within the Petri net formalism to deal with more complex and flexible split/join constructs.

The process model is used for generating an event log with 1000 random traces. This log is employed to illustrate the different mining steps of the FHM. Afterwards, this event log is adopted to generate others with 5%, 10% and 20% noise. To incorporate noise in the event logs we define five different types of noise generating operations: (i) delete the head of a trace, (ii) delete the tail of a trace, (iii) delete a part of the body, (iv) remove one event, and finally (v) interchange two random chosen events. During the deletion-operations at least one event, and no more than one third of the trace, is deleted. To incorporate noise, the traces of the original noise-free event log are randomly selected and then one of the five above described noise generating operations is applied (each noise generation operation with an equal probability of 1/5). The resulting noisy event logs are used in Subsection 5.1 to illustrate the mining behavior of the FHM in combination with noise. The combination of parallelism (after task A two parallel processes are started), loops (length-one, length-two and longer loops), hidden tasks, low-frequent behavior, and noise, make this event log difficult to mine.

As indicated before, process models in the FHM approach are not Petri nets but so-called "Causal nets" (C-nets). Next, we will first define the concept of a C-net and illustrate the concept by the translation of the Petri net in Fig. 2 into a C-net.

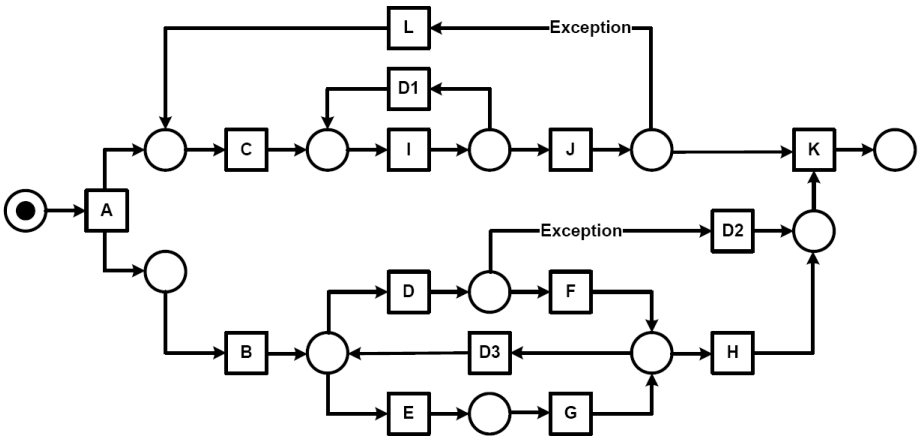

Fig. 2. The Petri net process model used as reference for generating event logs (with and without noise). The hidden tasks $D1$, $D2$, and $D3$ are not registered in the event-logs.

3 Internal Representation

Definition 1 (Causal net (C-net)). *A Causal net is a tuple* (T, I, O)*, where*

- T *is a finite set of tasks,*
- $I : T \to \mathcal{P}(\mathcal{P}(T))$ *is the input pattern function,*[1]
- $O : T \to \mathcal{P}(\mathcal{P}(T))$ *is the output pattern function.*

If $e \in T$ then $\Box e = \bigcup I(e)$ denotes the input tasks of e and $e\Box = \bigcup O(e)$ the output tasks of e.[2]

Definition 2 (Dependency Graph (DG)). *If C-net* $= (T, I, O)$ *is a Causal net then the corresponding Dependency Graph (DG) is a relation on* T *(*$DG \subseteq T \times T$*), with*

- $DG = \{(a, b) | (a \in T \wedge b \in a\Box) \vee (b \in T \wedge a \in \Box b)\}$

As an example, we show how the Petri net in Fig. 2 can be represented as a C-net (see Table 1). The Petri net in Fig. 2 has 12 tasks $(A, B, ..., L)$, so the corresponding task set $T = \{A, B, ..., L\}$.

I	$ACTIVITY$	O
$\{\}$	A	$\{\{B, C\}\}$
$\{\{A\}\}$	B	$\{\{E\}, \{D\}\}$
$\{\{A\}, \{L\}\}$	C	$\{\{I\}\}$
$\{\{B\}, \{F\}, \{G\}\}$	D	$\{\{F\}, \{K\}\}$
$\{\{B\}, \{F\}, \{G\}\}$	E	$\{\{G\}\}$
$\{\{D\}\}$	F	$\{\{D\}, \{E\}, \{H\}\}$
$\{\{E\}\}$	G	$\{\{D\}, \{E\}, \{H\}\}$
$\{\{F\}, \{G\}\}$	H	$\{\{K\}\}$
$\{\{C\}, \{I\}\}$	I	$\{\{I\}, \{J\}\}$
$\{\{I\}\}$	J	$\{\{K\}, \{L\}\}$
$\{\{J, H\}, \{J, D\}\}$	K	$\{\}$
$\{\{J\}\}$	L	$\{\{C\}\}$

Table 1. The translation of the Petri net (Fig. 2) into a C-net.

For each task the table shows an input (I) and an output (O) set expression. The set of subsets in the I-column describes which subsets of tasks should occur to enable the occurrence of the given task at the middle column. Tasks in the same subset are in the logical *and*-relation. The subsets themselves are in an *or*-relation. For instance, consider task H in Fig. 2. This task can occur whenever task F *or* G occurs. So, $I(H) = \{\{F\}, \{G\}\}$. Similarly, the set expressions in the

[1] $\mathcal{P}(X)$ denotes the powerset of some set X.

[2] $\bigcup I(e)$ is the union of the subsets in $I(e)$. For instance if $I(K) = \{\{J, H\}, \{J, D\}\}$ then $\Box K = \{J, H, D\}$.

O-column shows which tasks may be executed after the execution of a given task. For instance, consider task A in Fig 2. Since both tasks B *and* C are executed after the execution of A, $O(A) = \{\{B, C\}\}$. Remark that the set expressions can be straightforwardly translated into logical expressions. The input set expression $\{\{J, H\}, \{J, D\}\}$ of task K can thus be seen as the same as the logical expression $(J \wedge H) \vee (J \wedge D)$.

4 The FlexibleHeuristicsMiner(FHM) algorithm

To construct a process model on the basis of an event log, the log should be analyzed for causal dependencies, e.g., if a task is always followed by another task it is likely that there is a dependency relation between both tasks. To analyze these relations we first build a dependency graph (DG). The building of the DG in FHM is exactly the same as in the HM [10]. For the sake of completeness the necessary basic relations, measures and the construction of the DG (Subsection 4.1) are again presented in this paper.

Definition 3 (Basic Relations). *Let T be a set of tasks. $\delta \in T^*$ is a process trace, $W : T^* \to \mathcal{N}$ is a event log*[3]*, and $a, b \in T$:*

1. *$a >_W b$ iff there is a trace $\delta = t_1 t_2 t_3 \ldots t_n$ and $i \in \{1, \ldots, n-1\}$ such that $\delta \in W$ and $t_i = a$ and $t_{i+1} = b$ (direct successor),*
2. *$a >>_W b$ iff there is a trace $\delta = t_1 t_2 t_3 \ldots t_n$ and $i \in \{1, \ldots, n-2\}$ such that $\delta \in W$ and $t_i = t_{i+2} = a$ and $t_{i+1} = b$ and $a \neq b$ (length-two loops),*
3. *$a >>>_W b$ iff there is a trace $\delta = t_1 t_2 t_3 \ldots t_n$ and $i < j$ and $i, j \in \{1, \ldots, n\}$ such that $\delta \in W$ and $t_i = a$ and $t_j = b$ (direct or indirect successor).*

4.1 Step 1: Mining of the dependency graph (DG)

The starting point of the FHM is the construction of a so-called *dependency graph* (DG). A frequency-based metric is used to indicate how certain we are that there is a truly dependency relation between two events A and B (notation $A \Rightarrow_W B$). The calculated $\Rightarrow_W$ values between the events of an event log are used in a heuristic search for the correct dependency relations.

Definition 4 (Dependency measures). *Let W be an event log over T, $a, b \in T$, $|a >_W b|$ the number of times $a >_W b$ occurs in W, and $|a >>_W b|$ is the number of times $a >>_W b$ occurs in W.*[4]

$$a \Rightarrow_W b = \left(\frac{|a >_W b| - |b >_W a|}{|a >_W b| + |b >_W a| + 1} \right) if \quad (a \neq b) \tag{1}$$

[3] T^* is the set of all sequences (i.e., traces) that are composed of zero or more tasks of T. $W : T^* \to \mathcal{N}$ is a function from the elements of T^* to $\mathcal{N}$ (i.e., the number of times an element of T^* appears in the process log). In other words, W is a bag of traces.

[4] Because the event log W is a bag, the same trace can appear more than once in the log and patterns can appears more times in a trace. If, for instance, the pattern ab appears twice in a trace (e.g., c**ab**efgc**ab**efh), and this trace appears three times in W (i.e., W(c**ab**efgc**ab**efh)=3) then these appearances count as 6 in the $|a >_W b|$ measurement.

$$a \Rightarrow_W a = \left(\frac{|a >_W a|}{|a >_W a| + 1} \right) \tag{2}$$

$$a \Rightarrow^2_W b = \left(\frac{|a >>_W b| + |b >>_W a|}{|a >>_W b| + |b >>_W a| + 1} \right) \tag{3}$$

First, remark that the value of $a \Rightarrow_W b$ is always between -1 and 1. Some simple examples demonstrate the rationale behind this definition. If we use this definition in the situation that, in 5 traces, task A is directly followed by task B but the other way around never occurs, the value of $A \Rightarrow_W B = 5/6 = 0.833$ indicates that we are not completely sure of the dependency relation (only 5 observations possibly caused by noise). However, if there are 50 traces in which A is directly followed by B but the other way around never occurs, the value of $A \Rightarrow_W B = 50/51 = 0.980$ indicates that we are pretty sure of the dependency relation. If there are 50 traces in which task A is directly followed by B and noise caused B to follow A once, the value of $A \Rightarrow_W B$ is $49/52 = 0.94$ indicating that we are still pretty sure of a dependency relation.

A high $A \Rightarrow_W B$ value strongly suggests that there is a dependency relation between tasks A and B. We can use the dependency measures of Definition 4 in two different ways: (i) directly (i.e., without the *all-tasks-connected* heuristic), and (ii) in combination with the *all-tasks-connected* heuristic.

Without the use of the *all-tasks-connected* heuristic three threshold parameters are available in the FHM to indicate that we will accept a dependency relation: (i) the *Dependency threshold*, (ii) the *Length-one loops threshold* and (iii) the *Length-two loops threshold*. Usually the three parameters (i.e., *the Dependency thresholds*) have the same value (default 0.9). However, by using different parameters it is, for instance, possible to build a model without length-one loops (choose the *Length-one loops threshold* = 1.0). With these thresholds we can indicate that we accept dependency relations between tasks that have a dependency measure above the value of the dependency thresholds resulting in a control-flow model with only the most frequent tasks and behavior. By changing the parameters we can influence how complete the control-flow model becomes.

The advantage of using the *all-tasks-connected* heuristic is that many dependency relations are tracked without any influence of any parameter setting. The result is a relative complete and understandable control-flow model even if there is some noise in the log. The underlying intuition in the *all-tasks-connected* heuristic is that each non-initial task must have at least one other task that is its cause, and each non-final task must have at least one dependent task. Using this information we can first build a work flow model taking *the best* candidates (i.e., with the highest $A \Rightarrow_W B$ scores). One extra parameter is available in combination with the *all-tasks-connected* heuristic the so-called *relative to best threshold*. With this threshold we can indicate that we will also accept dependency relations between tasks that have (i) a dependency measure above the value of the *dependency threshold*, and (ii) have a dependency measure "close" to the first already accepted dependency value (i.e., for which the difference with the "best" dependency measure is lower than the value of *relative-to-best threshold*). However, if we use this heuristic in the context of a low-structured process the result is a very

complex model with all tasks and a high number of connections (as indicated in Fig 1).

In the next Sections the details of the *all-tasks-connected* heuristic are given. The all-tasks-connected heuristic is implemented in the algorithm items 4 through 9. In the items 10, 11, and 12 the minimal connected process model is extended with other reliable connections.

For practical reasons, we start adding two artificial tasks to identify univocally the beginning and the end of the process. This is especially practical if there is not a clear unique *start* and *end* task (e.g., if there is noise in the event log).

Definition 5 (Start/end extension). *Let W be an event log over T. Then W^+ is the (artificial) start/end-extension over T^+ with*

1. $T^+ = T \cup \{start, end\}$
2. $W^+ = \{start\ \delta\ end \mid \delta \in W\}$

Definition 6 (Dependency Graph (DG)-algorithm). *Let W be an event log over T, W^+ an event log over T^+ (i.e., the start/end-extension of W), σ_a the (absolute) Dependency Threshold (default 0.9), σ_{L1L} the Length-one-loops Threshold (default 0.9), σ_{L2L} the Length-two-loops Threshold (default 0.9), and σ_r the Relative-to-best Threshold (default 0.05). $DG(W^+)$ (i.e., the dependency graph for W^+) is defined as follows.*

1. $T = \{t \mid \exists_{\sigma \in W^+}[t \in \sigma]\}$ *(the set of tasks appearing in the log),*
2. $C_1 = \{(a,a) \in T \times T \mid a \Rightarrow_W a \geq \sigma_{L1L}\}$ *(length-one loops),*
3. $C_2 = \{(a,b) \in T \times T \mid (a,a) \notin C_1 \wedge (b,b) \notin C_1 \wedge a \Rightarrow_{2\ W} b \geq \sigma_{L2L}\}$ *(length-two loops),*
4. $C_{out} = \{(a,b) \in T \times T \mid b \neq End \wedge a \neq b \wedge \forall_{y \in T}[a \Rightarrow_W b \geq a \Rightarrow_W y]\}$ *(for each task, the strongest follower),*
5. $C_{in} = \{(a,b) \in T \times T \mid a \neq Start \wedge a \neq b \wedge \forall_{x \in T}[a \Rightarrow_W b \geq x \Rightarrow_W b]\}$ *(for each task, the strongest cause),*
6. $C'_{out} = \{(a,x) \in C_{out} | (a \Rightarrow_W x) < \sigma_a \wedge \exists_{(b,y) \in C_{out}} [(a,b) \in C_2 \wedge ((b \Rightarrow_W y) - (a \Rightarrow_W x)) > \sigma_r]\}$ *(the weak outgoing-connections for a length-two loop),*
7. $C_{out} = C_{out} - C'_{out}$ *(remove the weak connections),*
8. $C'_{in} = \{(x,a) \in C_{in} | (x \Rightarrow_W x) < \sigma_a \wedge \exists_{(y,b) \in C_{in}} [(a,b) \in C_2 \wedge ((y \Rightarrow_W b) - (x \Rightarrow_W a)) > \sigma_r]\}$ *(the weak incoming-connections for a length-two loop),*
9. $C_{in} = C_{in} - C'_{in}$ *(remove the weak connections),*
10. $C''_{out} = \{(a,b) \in T \times T \mid (a \Rightarrow_W b) \geq \sigma_a \vee \exists_{(a,c) \in C_{out}} [((a \Rightarrow_W c) - (a \Rightarrow_W b)) < \sigma_r]\}$,
11. $C''_{in} = \{(b,a) \in T \times T \mid (b \Rightarrow_W a) \geq \sigma_a \vee \exists_{(b,c) \in C_{in}} [((b \Rightarrow_W c) - (b \Rightarrow_W a < \sigma_r]))\}$,
12. $DG = C_1 \cup C_2 \cup C''_{out} \cup C''_{in}$.

To illustrate the algorithm as given above we apply the DG-algorithm on the event log generated with the process model as given in Fig. 2. As noticed before, the hidden tasks D1, D2 and D3 are not registered. The basic information we will use is in Table 2 (the counting of the direct successors (i.e., $a >_w b$)), Table 3 (the

counting of the length-two loops (i.e., $a >>_w b$)), and Table 4 (the dependency measures).

	Start	A	B	C	D	E	F	G	H	I	J	K	L	End
Start	0	1000	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	520	480	0	0	0	0	0	0	0	0	0	0
B	0	0	0	360	182	198	0	0	0	233	27	0	0	0
C	0	0	338	0	125	128	40	48	8	349	0	0	0	0
D	0	0	0	63	0	0	586	0	0	193	68	5	6	0
E	0	0	0	73	0	0	0	619	0	236	67	0	3	0
F	0	0	0	16	124	134	0	0	327	212	88	0	7	0
G	0	0	0	16	143	145	0	0	359	220	105	0	10	0
H	0	0	0	11	0	0	0	0	0	252	105	614	5	0
I	0	0	119	0	209	236	179	210	166	315	576	0	0	0
J	0	0	23	0	135	155	102	117	118	0	0	381	5	0
K	0	0	0	0	0	0	0	0	0	0	0	0	0	1000
L	0	0	0	17	3	2	1	4	9	0	0	0	0	0
End	0	0	0	0	0	0	0	0	0	0	0	0	0	0
#	1000	1000	1000	1036	921	998	908	998	987	2010	1036	1000	36	1000

Table 2. Direct successor ($a >_w b$-counting) and frequency (last line) counting.

1. The first step of the algorithm is the construction of the set A (the set of all tasks appearing in the log).
2. Looking at the diagonal of Table 2 there is only one candidate for C_1: task I is 315 times followed by itself. The value of $I \Rightarrow_W I = 315/(315+1) \geq \sigma_{L1L}$, resulting in $C_1 = \{(I, I)\}$.
3. For this step of the algorithm we make use of Table 3. The table indicates that pattern DFD appears 89 times in the log and pattern FDF 110 times. Therefore $D \Rightarrow_{2\,W} F = (89+110)/(89+110+1) = 0.995$. Because $F \notin C_1$ and $D \notin C_1$ and $0.995 \geq \sigma_{L2L}$ both $(F, D) \in C_2$ and $(D, F) \in C_2$. The same argumentation counts for the pattern EG resulting in $C_2 = \{(F, D), (D, F), (E, G), (G, E)\}$.
4. Based on Table 4 check each non *End*-row for the highest value (the strongest follower). For example, for the C task the highest value (in boldface) is 0.997; therefore (C, I) is in the set C_{out}.
5. Based on Table 4 check each non *Start*-column for the highest value (the strongest cause). For example, for the K task the highest value (in boldface) is 0.998; therefore (H, K) is in the set C_{in}.

6,7. As an illustration we take the tasks D and F. They are in a direct loop (i.e., $(D, F) \in C_2$). The strongest output connection of D beside F is K (0.833), and from F is H (0.997). For this reason $(D, K) \in C'_{out}$ (is not strictly necessary) and will be removed from C_{out} (step 7 of the algorithm). In Table 4 the removed connections are marked with underlining.

8,9. Analogue to step 6 and 7, but now for the incoming connections.

	Start	A	B	C	D	E	F	G	H	I	J	K	L	End
Start	0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	0	0	0	0	0	0	0	0
B	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C	0	0	0	0	0	0	0	0	0	0	0	0	0	0
D	0	0	0	0	0	0	89	0	0	0	0	0	0	0
E	0	0	0	0	0	0	0	104	0	0	0	0	0	0
F	0	0	0	0	110	0	0	0	0	0	0	0	0	0
G	0	0	0	0	0	133	0	0	0	0	0	0	0	0
H	0	0	0	0	0	0	0	0	0	0	0	0	0	0
I	0	0	19	0	40	63	59	57	97	116	0	0	0	0
J	0	0	0	0	0	0	0	0	0	0	0	0	0	0
K	0	0	0	0	0	0	0	0	0	0	0	0	0	0
L	0	0	0	0	0	0	0	0	0	0	0	0	0	0
End	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 3. Length-two loops counting ($a >>_w b$-counting). The value 89 in position D,F indicates that there are 89 DFD patterns in the event log. Remark the high value between L1L-task I and many other tasks (i.e., B, D, E F, G and H). This is caused by the looping behavior of I in combination with the parallel behavior of the other mentioned tasks.

	Start	A	B	C	D	E	F	G	H	I	J	K	L	End
Start	0	**.999**	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	**.998**	**.998**	0	0	0	0	0	0	0	0	0	0
B	0	0	0	.031	**.995**	**.995**	0	0	0	.323	.084	0	0	0
C	0	0	0	0	.328	.272	.421	.492	0	**.997**	0	0	0	0
D	0	0	0	0	0	0	**.650**	0	0	0	0	**.833**	.300	0
E	0	0	0	0	0	0	0	**.620**	0	0	0	0	.167	0
F	0	0	0	0	0	*.993*	0	0	**.997**	.0842	0	0	.667	0
G	0	0	0	0	*.993*	0	0	0	**.997**	.0232	0	0	.400	0
H	0	0	0	.15	0	0	0	0	0	.205	0	**.998**	0	0
I	0	0	0	0	.040	0	0	0	0	0	**.998**	0	0	0
J	0	0	0	0	.328	.395	.073	.054	.058	0	0	**.997**	**.833**	0
K	0	0	0	0	0	0	0	0	0	0	0	0	0	**.999**
L	0	0	0	**.944**	0	0	0	0	.267	0	0	0	0	0
End	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 4. All positive $a \Rightarrow_W b$-values. See the example for a clarification of the bold face, Italic and underlined values.

10,11 Depending on the values of the parameter settings, extra connections are accepted if the absolute dependency threshold σ_a (default 0.9) or the relative-to-best threshold σ_r (default 0.05) is fulfilled. Remark that for the default parameter setting the dependency relation between D and K is not accepted because $D \Rightarrow_W K = 0.333 < 0.9$ (Table 4). However, the connection from J to L is accepted, because the *all tasks connected* heuristic is active. In the matrix of Table 4 the extra accepted dependency values are displayed in *Italics*.

12. Finally we can combine the information in the different matrices to perform the last step of the algorithm.

$\Box X$	$ACTIVITY$	$X\Box$
$\{\}$	A	$\{B, C\}$
$\{A\}$	B	$\{D, E\}$
$\{A, L\}$	C	$\{I\}$
$\{B, F, G\}$	D	$\{F\}$
$\{B, F, G\}$	E	$\{G\}$
$\{D\}$	F	$\{D, E, H\}$
$\{E\}$	G	$\{D, E, H\}$
$\{F, G\}$	H	$\{K\}$
$\{C, I\}$	I	$\{I, J\}$
$\{I\}$	J	$\{K, L\}$
$\{J, H\}$	K	$\{\}$
$\{J\}$	L	$\{C\}$

Table 5. The resulting DG in table layout.

If we compare Table 5 with the result of applying Definition 2 on the C-net as given in Table 1 the only difference is the missing low frequent connection from D to K. A graphical representation (ProM 6.0) of the same result is given in Fig. 3. This graph is augmented with extra information. The numbers in the task boxes indicate the frequency of the task; the numbers on the arcs indicate the reliability of the dependency relation. Other views are also possible within ProM. As indicated, the low frequent connection from D to K is missing in the DG. However, If we use the parameter settings $\sigma_a = 0.80$ and $\sigma_r = 0.20$ the low frequent connection from D to K is also accepted (the frequency of this connection is only 13). The resulting (complete) graph, but now in combination with frequency information for the arcs, is given in Fig. 4. Remark that if we use the *all-tasks-connected* heuristic, all tasks (also low frequent tasks like tasks L) will be part of the mined model. If we do not like to have low frequent tasks in our model it is possible to use the FHM without the *all-tasks-connected* heuristic or to use one of the ProM filters to remove low frequent tasks out of the event log.

The DG gives information about the dependency between tasks, but the types of splits/joins are not yet mined. This mining is the subject of the next subsection.

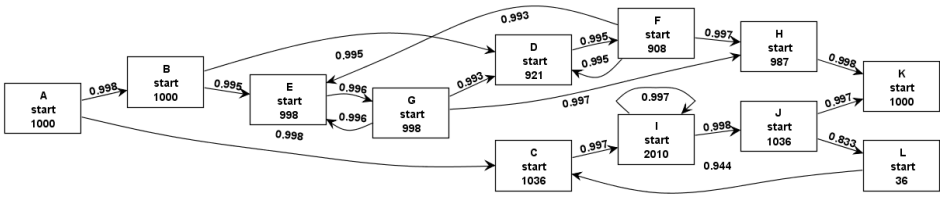

Fig. 3. The resulting dependency graph (DG) with dependency information if we use the default parameters setting. The low frequent connection between taks D and K is not in the model.

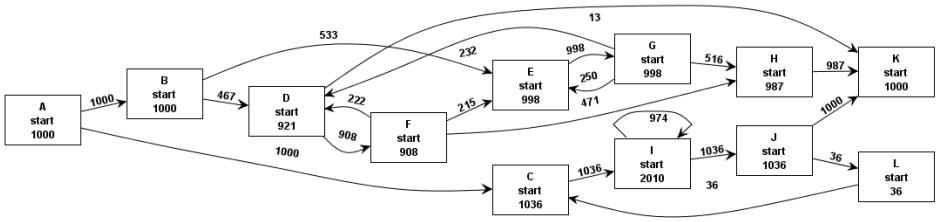

Fig. 4. The resulting dependency graph (DG) with frequency information if we use the parameter settings $\sigma_a = 0.80$ and $\sigma_r = 0.20$. The low frequent connection (i.e., 13 cases) from D to K is also accepted.

4.2 Step 2: mining of the splits/joins

The next step of the FlexibleHeuristicsMiner is the characterization of *split* and *join* points of the DG. Thus, for each task in the DG, the different split and join patterns are mined. Let we first explain the basic idea. Starting with task A of the DG of Table 5 the output set is $\{B, C\}$. However, we want to know whether task A is always followed by both B and C (i.e., an AND-split), only by either B or C (i.e., a XOR-split), or most of time by B or C and sometimes by both (i.e., an OR-split). We will use a simple extension of the C-net formalism (Definition 1) to characterize the behavior of the splits and joins. The mining of the splits/joins mainly relies on two data structures: (i) the DG and (ii) the event log that contains information about the ordering of the tasks. The result is an *augmented-C-net*. The augmented-C-net is an C-net but with bags instead of sets so that it becomes possible to indicate the number of times specific split and join patterns appear in the event log. This information is the basis for statistical computing of valid splits/joins.

Definition 7 (augmented Causal net (augmented-C-net)). *An augmented Causal net is a tuple (T, I, O), where*

- *T is a finite set of tasks,*
- *$I : T \to \mathcal{P}(\mathcal{P}(T) \to \mathcal{N})$ is the input frequency function,*
- *$O : T \to \mathcal{P}(\mathcal{P}(T) \to \mathcal{N})$ is the output frequency function.*

Based on the information in the event log it appears that task A (frequency 1000) is always followed by both B and C. In the augmented-C-net (Table 6) this is indicated by the output-*bag* of task A (i.e., $O(A) = \{\{B, C\}^{1000}\}$). The output bag of task B (i.e., $O(B) = \{\{D\}^{533}, \{E\}^{467}\}$ is an example of a XOR-split.

I	$TASK$	O
$[]$	A	$[\{B, C\}^{1000}]$
$[\{A\}^{1000}]$	B	$[\{D\}^{467}, \{E\}^{533}]$
$[\{A\}^{1000}, \{L\}^{36}]$	C	$[\{I\}^{1036}]$
$[\{B\}^{467}, \{F\}^{222}, \{G\}^{232}]$	D	$[\{F\}^{908}, \emptyset^{13}]$
$[\{B\}^{533}, \{F\}^{215}, \{G\}^{250}]$	E	$[\{G\}^{998}]$
$[\{D\}^{908}]$	F	$[\{D\}^{222}, \{E\}^{215}, \{H\}^{471}]$
$[\{E\}^{998}]$	G	$[\{D\}^{232}, \{E\}^{250}\{H\}^{516}]$
$[\{F\}^{471}, \{G\}^{516}]$	H	$[\{K\}^{987}]$
$[\{C\}^{1036}, \{I\}^{974}]$	I	$[\{I\}^{974}, \{J\}^{1036}]$
$[\{I\}^{1036}]$	J	$[\{K\}^{1000}, \{L\}^{36}]$
$[\{J, H\}^{987}, \{J\}^{13}]$	K	$[]$
$[\{J\}^{36}]$	L	$[\{C\}^{36}]$

Table 6. The augmented-C-net for the DG of Table 5 in combination with the event log with 1000 traces.

In the ProM implementation of FHM another visualization of augmented-C-net is used. By clicking a task (e.g., task F) in the DG graph (Fig. 3) the split and join information of that task is displayed (Fig. 5).

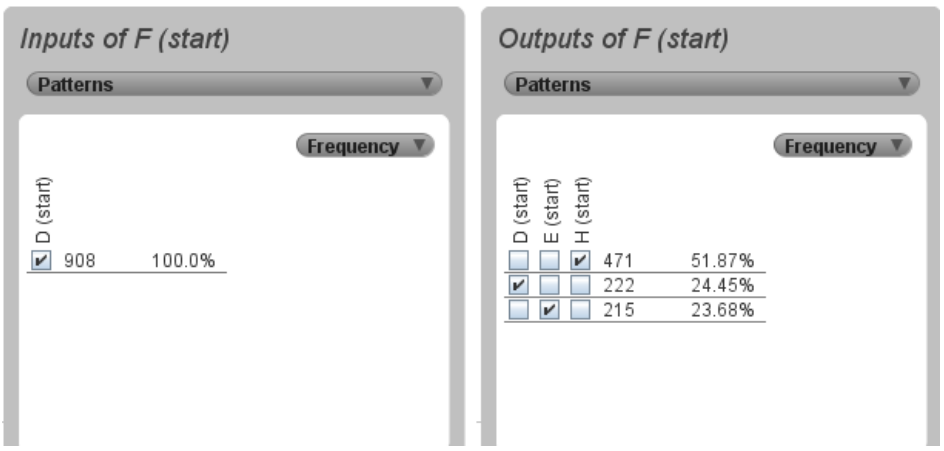

Fig. 5. The split and join information after clicking tasks F in the DG of Fig. 3. Each line corresponds to a pattern in which the activated outputs are identified by the '✓' symbol.

The basic idea behind the building of the augmented-C-net is relatively simple. We take task A, the DG of Table 5, and the trace $ABDCIFIJEGHK$ as example. We first look at the split information for A. Because $A\Box = \{B, C\}$, $\Box B = \{A\}$ and $\Box C = \{A, L\}$ (see the DG in Table 5) we know that there are two candidates that can be activated by A. Because both tasks B and C appear in the trace and A is the nearest candidate appearing before B and C, we take the position that both B and C are activated by the current A and the split frequency information of A is updated with the pattern $\{B, C\}$ (i.e., $O(A) = O(A) \biguplus [\{B, C\}]$).

However, more complex situations are possible. For instance look at the split of tasks B. $B\Box = \{D, E\}$ and $\Box D = \Box E = \{B, F, G\}$. That means that there are three candidate tasks for the activation of D and E. If we look at the trace $ABDCIFIJEGHK$ the only available candidate for D is B (i.e., B is the only candidate that appears before D). For E there are two candidates B and F both appearing before E. Because the distance between F and E is closer than the distance between B and E we take the position that F is the activator of E.[5] Therefore the split frequency information of B is updated with the pattern $\{D\}$ (i.e., $O(B) = O(B) \biguplus [\{D\}]$).

Finally we look at the split I in combination with the first appearance of I in the trace $ABDCIFIJEGHK$. $I\Box = \{I, J\}$, $\Box I = \{CI\}$ and $\Box J = I$. In other words task I can activate I and J. The only candidate for the second I in the trace is the first I. Based on the nearest candidate strategy we take the position that task J is caused by the second appearance of I. In other words the split information for the first appearance of I results in updating of the output frequency with $\{I\}$ (i.e., $O(I) = O(I) \biguplus [\{I\}]$). When there is only one candidate we will take this candidate as the activator.

For the mining of the frequency information of the joins of the DG we follow the same strategy but now we go backwards through the traces. Table 6 shows the resulting augmented-C-net.

Remark that an augmented-C-net contains both work-flow information and specific frequency information as emerging in the event log. In Subsection 5.3 we will illustrate how this combined information can be used for understandable conformance checking. Remark also that an augmented-C-net can easily be translated into the corresponding C-net or in a simplified C-net (i.e., by only representing the high-frequent patterns into the C-net).

4.3 Step 3: mining long-distance dependencies

The final step of the FlexibleHeuristicsMiner is the identification of dependencies that are not represented yet in the DG. Called long-distance dependencies (or non-free choice), these relations indicate cases in which a task X depends indirectly on another task Y to be executed. That means that, in a split or join point, the choice may depend on choices made in other parts of the process model. Fig. 6 depicts a Petri Net with two distinct long-distance dependencies (i.e., the relations $B \Rightarrow E$

[5] We only take tasks appearing before E as possible candidates. The choice of the nearest candidate is only one of the possible selection strategies.

and $C \Rightarrow F$). Note that, in this example, there are only two possible sequences: $ABDEG$ and $ACDFG$. However, without mining the long-distance dependencies, the DG does not ensure that task E is only executed if D follows B. The same happens for F. Thus, non-valid sequences such as $A\underline{B}D\underline{F}G$ or $A\underline{C}D\underline{E}G$ might fit in the process model. Fig. 7 shows the DG without mining the long-distance dependencies.

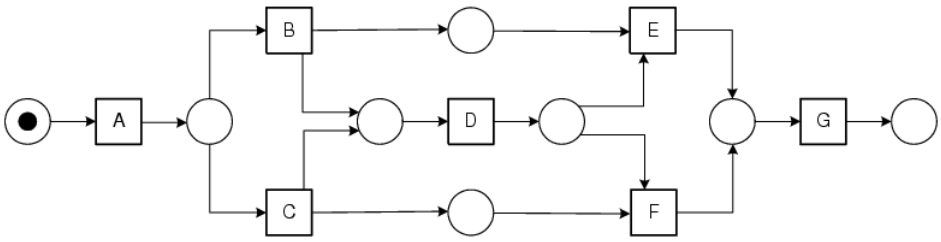

Fig. 6. A process model (in the Petri net formalism) with a long-distance dependency construct.

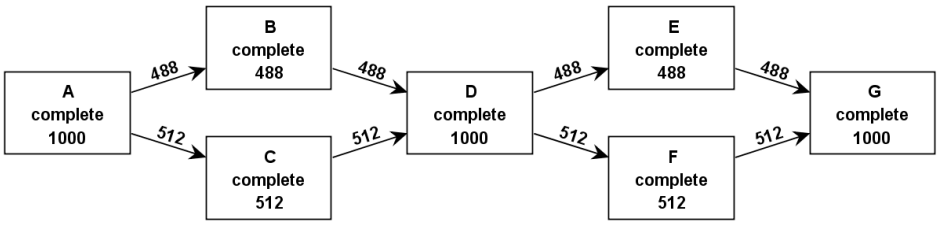

Fig. 7. The Fig. 6's corresponding DG without long-distance dependency relations.

In order to handle the long-distance dependency issue, a new frequency-based metric is defined. Basically, this metric takes into account the indirect relation between tasks (i.e., the direct or indirect successor counter of Definition 3). The main idea is to find pairs of tasks with similar frequencies in which the first task is directly or indirectly followed by the second one. These circumstances are measured through the $a \Rightarrow^{l}_{W} b$ measure (Definition 8). All the pairs with high $\Rightarrow^{l}_{W}$-values (i.e., close to 1) are designated as long-dependency relations.

Definition 8 (Long distance dependency measure). *Let W be an event log over T, $a, b \in T$, $|a >>>_W b|$ the number of times $a >>>_W b$ occurs in W*[6]*, and*

[6] In the pattern cde**a**fg**b**hi**b**jk**a**l**a**def**b**gh only the underlined appearances of the pattern a...b contribute to the value $|a >>>_W b|$ (i.e., only a...b patterns without other a's or b's between them).

$|a|$ *is the number of times* a *occurs in* W.

$$a \Rightarrow^{l}_{W} b = \left(\frac{2\ (|a >>>_W b|)}{|a| + |b| + 1}\right) - \left(\frac{2\ Abs(|a| - |b|)}{|a| + |b| + 1}\right) \tag{4}$$

A value close to 1 of the first part of the expression indicates that task a is always followed by task b. A value close to 0 of the second part indicates that the frequency of tasks a and b is about equal. That means that an overall value close to 1 indicates both: task a is always followed by task b and the frequencies of tasks a and b are about equal[7]. Remark that some long-dependency cases are already indirectly represented in the DG. A good example is the relation $A \Rightarrow D$ in Fig. 6, which its long-distance dependency value is close to 1.0 but no extra dependency relation is necessary. This happens because A is always indirectly followed by D, turning redundant the extra direct connection from A to D. With this remark, it is finally defined that a long-dependency relation $X \Rightarrow Y$ (with $X, Y \in T$) needs a new dependency relation in the DG whenever $X \Rightarrow^{l}_{W} Y \geq \sigma_l$ (σ_l is a long-distance threshold; by default $\sigma_l = 0.90$) and it is possible to go from X to the end task without visiting Y. Note that every time a new (long-distance) dependency relation is added into the DG the relation tasks' inputs and outputs change as well as the split/join points. So, at the end of this stage (mining long-distance dependencies), it is necessary to recompute the split/join information.

Up to here, the details of the process representation formalism and the mining algorithm are presented. In the next section we illustrate the mining results in case of noise and in case of a low-structured domain.

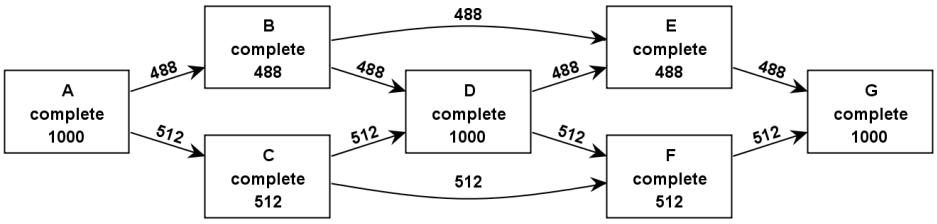

Fig. 8. The Fig. 6's corresponding DG with long-distance dependency relations.

[7] The requirement that the frequency of both tasks B and E are roughly equal is a restriction that brings about that not all possible long-distance dependency relations are mined.

5 Noise, Low Structured Domains, and Conformance Checking

5.1 Noise

In a first experiment we illustrate the usage of the FHM on event logs with noise (i.e., the event logs with 5%, 10% and 20% noise as described in the Section 2). Both the effects during the mining of the DG (step 1 of the mining algorithm) and during the mining of augmented-C-net (step 2) are discussed.

First, the mining results at the DG level. We compare the mining results of the event log with noise with the mining results of the noise free event log (Fig. 3). Using the default parameter settings in combination with *all-tasks-connected* heuristic on the event logs with different noise levels, the same 19 dependency relations were successfully mined. The only difference is related to the dependency measures; the noise will cause a decrease in the dependency measures. Table 7 shows the differences in the average dependency measures (i.e., 19 relations) for the different noise levels.

	0%	5%	10%	20%
Average Dependency	0.9849	0.9829	0.9821	0.9814

Table 7. Evolution of average dependency measures in the DG for different noise levels.

In Subsection 4.1 we illustrated that it is possible to rediscover the complete dependency graph (inclusive the low frequent connection from tasks D to K) if we use the parameter settings $\sigma_a = 0.80$ and $\sigma_r = 0.20$ (Fig. 4). However, mining low frequent behavior in the situation were the event log contains noise is more problematic. By using low thresholds to accept also low frequent behavior, there is always the risk that we will also accept noise in the mined model. For instance, if we use the FHM with the same parameter settings as given above ($\sigma_a = 0.80$ and $\sigma_r = 0.20$) we will end up with the DG as given in Fig. 9. If we compare this graph with the DG's of Fig. 3 and Fig. 4 we can see that beside the low frequent connection between tasks D and K also two extra connections are introduced (i.e. from A to I and from F to K). These extra connections are based on behavior in the event log caused by noise. We can conclude that for the main behavior in the event log the impact of noise during the mining of the DG is almost negligible. However, the combination of an event log with noise and mining of detailed behavior appears obviously difficult.

The result of the first mining step is a DG. The result of applying mining step 2 on this DG in combination with an event log will result in an augmented-C-net. Unlike the mining of the DG, the mining of the augmented-C-net does not rely directly on any threshold. So, all the noisy information is represented in the augmented-C-net. The augmented-C-net can be transformed into a C-net. During this transformation we can take the decision to represented all or only the main behavior into the C-net. In this experiment we will use a threshold of 5%. Below

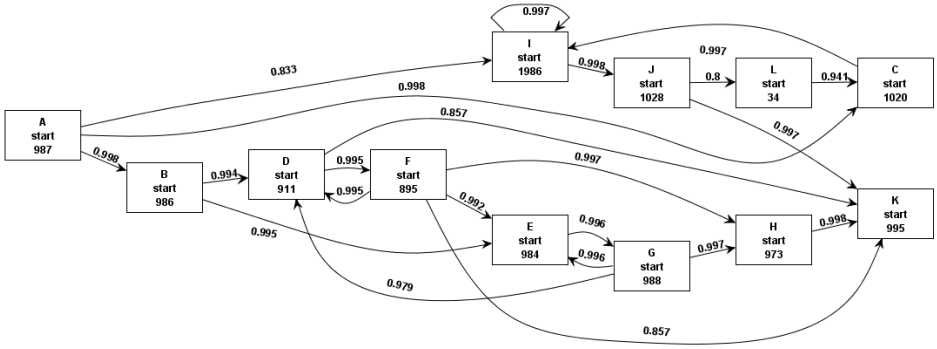

Fig. 9. The resulting dependency graph (DG) with dependency information if we use the event log with 10% noise in combination with the parameter settings $\sigma_a = 0.80$ and $\sigma_r = 0.20$. The low frequent connection from D to K but also the "wrong"-connections from A to I and from F to K are in the model.

the results of the augmented-C-net mining for the event logs with 5%, 10% and 20% noise are presented. Fig. 10 shows the augmented-C-net information of task F in the case of 20% noise. Comparing these results with the results in Fig. 5, it is possible to see that there is a clear distinction between the original patterns (the patterns with a high frequency) and the patterns caused by the noise (the patterns with a low frequency). Even for the experiment with the highest noise level (i.e., 20%), it is clear that the patterns caused by noise have frequency below a 5% threshold.

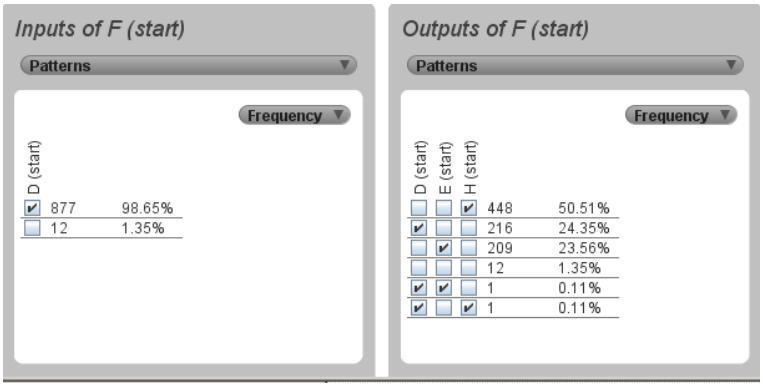

Fig. 10. The augmented-C-net information for task F in the case of 20% noise as presented in the ProM tool.

Table 8 shows the frequency of the patterns caused by noise for the different noise levels. For instance, the information about task F and with 20% noise is based on the corresponding split and join tables (Fig. 10). Thus, the final values

are JOINS = 1.35% ≈ 1.4% and SPLITS = 1.35%+0.11%+0.11% ≈ 1.6%. Therefore, it can be concluded that in this experimental setting, it is possible to recognize low frequent behavior in the split and join information, caused by noise.

TASK	*JOINS*			*SPLITS*		
	5%	10%	20%	5%	10%	20%
A				0.9%	1.6%	3.0%
B	0.2%	0.6%	0.7%	0.1%	0.5%	1.9%
C	0.2%	0.5%	0.7%	0.2%	0.6%	1.2%
D	0.4%	0.9%	1.3%	0.3%	1.0%	1.5%
E	0.4%	0.6%	1.4%	0.1%	0.5%	0.7%
F	0.2%	0.6%	1.4%	0.4%	1.0%	1.6%
G	0.3%	0.9%	1.3%	0%	0.2%	0.4%
H	0.4%	1.2%	1.6%	0.1%	0.1%	0.1%
I	0.4%	0.7%	1.2%	0.2%	0.4%	0.7%
J	0.2%	0.7%	1.2%	0.1%	0.5%	1.0%
K	0.3%	0.7%	1.4%			
L	0%	0%	0%	0%	0%	0%

Table 8. Quantity of low frequent patterns caused by noise with regard to the frequency of the task.

5.2 Low Structured Domains

The second part of this evaluation study is based on an event log from a low-structured domain. Using the event log introduced in Section 1, it is intended to show how the FHM can provide insight about very flexible applications. Having the example depicted by Fig. 1 as reference, we pretend to analyze the behavior a relevant task (identified at the right-hand side of the picture, with several incoming and outgoing connections). This analysis is done for two kinds of process models: (i) a *complete* model in which even the low-reliable dependency relations are considered, and (ii) a *simplified* model in which only the high-reliable dependency relations are taken into account. Note that, by space issues, this analysis is only done for the split patterns.

The FHM result for the complete model is depicted by Fig. 1. This model is characterized by a dense DG, which turns the model analysis in a difficult task. Nonetheless, there is a lot of information in the model that can be intuitively analyzed. A good example is the splits and joins characterization provided by the FHM. Table 9 shows how a given task behaves in this complex model. Note that each pattern presented in Table 9 has a corresponding bag expression. For instance, the first and the last patterns can be expressed by $\{O_8\}^{1151}$ and $\{O_7, O_8\}^{69}$. Additionally, the empty pattern that appears in the list (second one) is the result of two possible events: (i) the given task may be an end task, or (ii) the traces that originated those empty patterns are very specific cases (probably considered as noise) that do not fit in the DG.

OUTPUTS								*FREQUENCY*
O_1	O_2	O_3	O_4	O_5	O_6	O_7	O_8	
							✓	1151
								469
				✓				150
	✓							99
					✓			92
✓								71
						✓	✓	69
46 others								219

Table 9. The split patterns of the *B Perifeer infuus* task in the complete model.

The FHM result for the simplified model is depicted by Fig. 11. Contrarily the complete one, this model is characterized by a sparser DG. This means that the information provided by the DG is easier to understand by the analyst. However, since this sparse DG was obtained through abstraction processes, some of the information may be just omitted in the DG. Nevertheless, it is possible to identify these cases with the FHM using the splits and joins information of the augmented-C-net. Table 10 presents the splits and joins information for a given task of the simplified model.

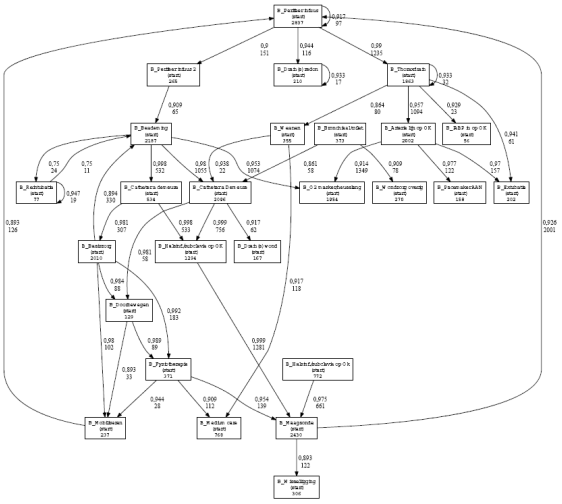

Fig. 11. The simplified control-flow mining result on an event log from a low-structured domain. This DG contains only relations with dependency value greater than 0.85. Table 10 contains the augmented-C-net split information of the upper node.

As expected, the patterns depicted in Table 10 are in line with the ones generated in the complete model. Although in the simplified model only four out

OUTPUTS				*FREQUENCY*
O_2	O_5	O_6	O_8	
			✓	1235
				550
	✓			151
✓				116
		✓		97
11 others				171

Table 10. The split patterns of the *B Perifeer infuus* task in the simplified model.

of eight outputs are taken into account, it is possible to characterize the main behavior of the given task. The reason why the very same pattern $\{O_8\}$ (the most frequent one) has a higher frequency in the simplified case is related with pattern combination. For instance, the complete case's patterns $\{O_8\}^{1151}$ and $\{O_7, O_8\}^{69}$ (and some other low frequent ones) are merged into the simplified case's pattern $\{O_8\}^{1235}$.

5.3 Conformance checking

In this subsection we have a closer look at the use of the augmented-C-net for conformance checking. During conformance checking we are interested in the conformance between a process model and an event log. The evaluation of the conformance between a model and a event log can take place in different dimensions: *fitness*, *precision*, and *generalization*. The fitness indicates how much of the **observed** behavior in the event log is captured by the process model. The precision indicates how much of **not observed** behavior in the event log can be recognized by using the process model. Generalization indicates wether the process model permit extra "allowed" behavior. Remark that the two first dimensions (i.e., fitness and precision) are a relation between a process model and an event log. In this subsection our focus is on these two dimensions. Note also the relation between a C-net and an augmented-C-net. The C-net is a process model, the augmented-C-net contains detailed information about the behavior as observed in an event log. For this reason it seems possible to use a comparison between the C-net (i.e., a process model) and the corresponding augmented-C-net (i.e, with event log information) to say something about fitness and precision. For instance if we translate an augmented-C-net into a simplified C-net by only representing high-frequent patterns in the C-net we know beforehand that not all observed behavior will fit in model. Below some other less trivial examples of the use of the augmented-C-net for conformance checking are given.

As a first example we have a closer look at the previous situation where we used the default parameters to mine the noise-free log. As indicated the resulting DG (Fig. 3) and the related augmented-C-net (Table 6) are both missing the connection between task D and K. That means that most of the traces of the event log fit exactly in the process model, only the traces that make use of the low frequent (i.e., 13 cases) dependency relation between tasks D and K will not

fit in the mined model (i.e., it is observed behavior that does not fit in the process model). The resulting augmented-C-net (Table 6) can be used to indicate places where the event log does not fit in the model. If we look at the output pattern of task D a deficiency is registered; there are 13 cases with an empty set as output pattern. In the input patterns of K the deficiency is indicated in the same table by 13 cases were only tasks J causes tasks K, and not both J and H. This means that we can use augmented-C-net to recognize missing or divergent behavior.

In the next example we will indicate how we can recognize "strange/not allowed" task sequences. To check if we can recognize extra behavior our starting point is an exact correct C-net model for the example event log without noise. Our goal is to indicate the extra behavior in the event log with 10% noise. To check this we first apply Def. 2 to get the associated DG. This is exactly the complete DG of Fig. 4. In the next step we use the event log with 10% noise in combination with step 2 of the FHM algorithm to build a new augmented-C-net. If we now compare the original C-net with the new augmented-C-net we can recognize the places in the C-net where there is extra/new behavior caused by the noise. This information is displayed in Table 11.

TASK	*JOINS*			*SPLITS*		
	5%	10%	20%	5%	10%	20%
A				9	16	29
B	2	6	7	1	5	10
C	2	5	7	2	6	12
D	4	8	12	0	4	9
E	4	6	14	1	5	7
F	2	5	12	4	9	14
G	3	9	13	0	2	4
H	4	12	15	1	1	1
I	8	14	23	3	7	13
J	2	7	12	1	5	10
K	9	24	37			
L	0	0	0	0	0	0
Totals	40	96	152	22	60	109

Table 11. Quantity of extra patterns in the augmented-C-net caused by the noise.

From Table 11 is clear that we can use the augmented-C-net to detect extra behavior. However, we can ask ourselves how much of the extra behavior is detected. Most of the noise operations will result in task patterns that are not allowed and are not in the original event log. The noise is based on five different types of noise generating operations: (i) delete the head of a trace, (ii) delete the tail of a trace, (iii) delete a part of the body, (iv) remove one event, and finally (v) interchange two random chosen events. In the case of 10% noise over 1000 traces we would expect 20 traces of each type. Type (i) and (ii) will possible generate

one new pattern in the frequency table. Type (iii) and (iv) two new patterns, and finally type (v) four new patterns. Therefore the maximal number of possible new patterns is 40 + 80 + 80 = 200. If we compare this with the totals for 10% noise situation in Table 11 the total number of observed "strange" behavior is 96 + 60 = 156. A possible explanation for this lower value is that not each noise operation will always result in a "not allowed" trace. For instance, interchanging two parallel tasks will not result in a "not allowed" trace.

In conclusion, the presented examples show that we can use augmented-C-net both for recognizing extra behavior and for recognizing missing behavior. Moreover, the augmented-C-net indicates in a very understandable way, where in the process model, these deviations are located.

6 Conclusions and Future Work

In this paper the basic ideas behind the flexible heuristic miner are presented: the development of a robust and flexible control-flow mining algorithm that performs well in practical situations and with results that are easy to understand. To achieve this goal two new process modeling formalism are introduced (i.e., Causal nets and augmented Causal nets). Also the three steps of the heuristics driven control-flow mining algorithm are defined (i.e., the Flexible Heuristics Miner (FHM)). A working example is used to illustrate the modeling formalism and the mining algorithm. Finally, the behavior of the FHM in situations with event logs with noise, event logs from low-structured domains, and the use of the augmented-C-net for conformance checking are illustrated. For the illustrative examples it appears possible to mine the main behavior in the event log and the approach seems robust for noise. However, there are still plenty challenges left.

To get a better understanding of the mining qualities of the FHM we have to perform more mining experiments with all kind of event logs (noise, complex and low structured domains, etc.). Also our claim that the resulting models of the FHM approach are easy to understand by the process owners needs some experimental evidence. The implementation of the long-distance mining is still incomplete; only simple long-distance dependencies can be mined. Also the use of the presented approach for understandable conformance checking needs supplementary experiments and research. Keeping the basic ideas, improvements of the mining algorithm seem possible.

Acknowledgments This work is being carried out as part of the project "Merging of Incoherent Field Feedback Data into Prioritized Design Information (DataFusion)", sponsored by the Dutch Ministry of Economic Affairs under the IOP IPCR program.

References

1. W.M.P. van der Aalst, B.F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A.J.M.M. Weijters. Workflow Mining: A Survey of Issues and Approaches. *Data and Knowledge Engineering*, 47(2):237–267, 2003.

2. A. Adriansyah, B.F. van Dongen, and W.M.P. van der Aalst. Towards robust conformance checking. In *Business Process Management (BPM 2010)*, volume xx of *Lecture Notes in Computer Science*, page xx. Springer-Verlag, Berlin, 2010.
3. B. van Dongen, A.K. Alves de Medeiros, H.M.W. Verbeek, A.J.M.M. Weijters, and W.M.P. van der Aalst. The ProM framework: A New Era in Process Mining Tool Support. In G. Ciardo and P. Darondeau, editors, *Application and Theory of Petri Nets 2005*, volume 3536 of *Lecture Notes in Computer Science*, pages 444–454. Springer-Verlag, Berlin, 2005.
4. C.W. Gunther. *Process Mining in Flexible Environments*. PhD thesis, Eindhoven University of Technology, Eindhoven, 2009.
5. R.S. Mans, M.H. Schonenberg, M.S. Song, W.M.P. van der Aalst, and P.J.M. Bakker. Application of process mining in healthcare: a case study in a Dutch hospital. *Communications in Computer and Information Science*, 25:425–438, 2009.
6. L. Maruster, W.M.P. van der Aalst, A.J.M.M. Weijters, A. van den Bosch, and W. Daelemans. Automated Discovery of Workflow Models from Hospital Data. In C. Dousson, F. Höppner, and R. Quiniou, editors, *Proceedings of the ECAI Workshop on Knowledge Discovery and Spatial Data*, pages 32–36, 2002.
7. A.K. Alves De Medeiros, A.J.M.M. Weijters, and W.M.P. van der Aalst. Genetic process mining : a basic approach and its challenges. *Data Mining and Knowledge Discovery*, 14(2):245–304, 2007.
8. A. Rozinat. *Process Mining: Conformance and Extension*. PhD thesis, Eindhoven University of Technology, Eindhoven, 2010.
9. A. Tiwari, C.J. Turner, and B. Majeed. A review of business process mining: state-of-the-art and future trends. *Business Process Management Journal*, 14(1):5–22, 2008.
10. A.J.M.M. Weijters, W.M.P. van der Aalst, and A.K. Alves de Medeiros. Process Mining with the HeuristicsMiner-algorithm. BETA Working Paper Series, WP 166, Eindhoven University of Technology, Eindhoven, 2006.