Unsupervised Hierarchical Process Mining with the Process Fragment Miner

Joern Tobis(✉), Felix Schumann, Juergen Mangler, and Stefanie Rinderle-Ma

TUM School of Computation, Information and Technology, Technical University of Munich, Garching, Germany
{joern.tobis,felix.schumann,juergen.mangler,
stefanie.rinderle-ma}@tum.de

Abstract. Abstracting processes into a hierarchical structure of subprocesses helps to improve the understandability and readability of process models. Mining such hierarchical process models from event logs is an active subfield in process mining research. Existing approaches often require activities to be labeled with a hierarchy notion or context data to identify hierarchies. In this work, we propose the ProcessFragmentMiner (PFM) to fragment an event log into subprocesses that represent the root process. PFM harnesses the dependency matrix generated by the heuristics miner in combination with different fragment ranking mechanisms. This work uses the inductive and split miner to mine the resulting process models for each fragment. PFM is evaluated against a supervised and an unsupervised hierarchical process mining approach from the literature. We find that PFM works best in combination with the presented Bigram ranking method and can match supervised approaches for some data sets. The proposed PFM approach enables hierarchical process mining on any event log without the need for any preprocessing steps.

Keywords: Process Mining · Automated Process Discovery · Hierarchical Process Discovery · Event Abstraction · Model Abstraction · Unsupervised Process Mining

1 Introduction

Structuring a process into subprocesses can increase the understandability of a process model [12] and is well established and supported by many modeling approaches, modeling tools, and execution engines, e.g., BPMN, Signavio, or CPEE [11]. Automatically discovering subprocesses and hierarchical structures from event logs is considered one approach to finding the right abstraction level for a process, which is an important problem in BPM [3]. The abstraction level at which events are reported is, in many cases, much lower than what a user actually requires from a process model [16]. Especially when dealing with larger processes in terms of tasks and process participants, finding a reasonable abstraction level to create a top-level representation of what is happening creates better understandability. Typical use cases span from automated production

C. Cappiello et al. (Eds.): CoopIS 2025, LNCS 15535, pp. 333–350, 2026.
https://doi.org/10.1007/978-3-032-15538-2_19

processes in manufacturing to more knowledge-driven processes, such as clinical processes. Solving this granularity problem can further foster the application of process mining, especially in knowledge-intensive processes [3].

Process mining approaches that deal with different abstraction levels are categorized by the taxonomy provided in [16]. A core distinction is the supervision strategy. Most approaches rely on a supervised approach, which either has a direct labeling of hierarchies, such as [10], or uses context data to identify labels, such as [9]. To overcome the need for a pre-labeled event log, we propose an unsupervised hierarchical mining approach called ProcessFragmentMiner (PFM). PFM enables an automatic organization of process activities into subprocesses. As input, the approach requires an event log, which is considered discrete event data. The unsupervised PFM approach brings multiple benefits to users of hierarchical process mining techniques. Being based purely on an event log, no additional labeling or preprocessing of events is needed. Since the process discovery part is only addressed in the last part of the pipeline, different discovery algorithms can be used.

The PFM approach can be understood as one step towards advancing process mining by "bridging the preprocessing (specifically, the event abstraction) and the Mining & Analysis phases of typical process mining workflows" [3]. The result of PFM is an initial hierarchical process model with a single hierarchy level that structures the process. Based on this, domain experts obtain an overview of the process at first sight and can take further refinement actions.

PFM is implemented as a five-step approach to mine hierarchical process models from an event log. The core of the approach is a dependency matrix as used by the HeuristicsMiner [15]. This dependency matrix is then used to calculate possible process subtraces. We propose three different ranking methods to rank subtraces before recombining them into a full process model. For each identified subprocess fragment, specific subprocess models are mined. Additionally, the process model between those fragments is mined. We evaluate the processes discovered by our approach based on established evaluation methods as presented in [2, 10] and compare them with the supervised domain knowledge approach and unsupervised random approach presented in [10].

The remainder of this article is structured as follows. In Sect. 2, we provide an overview of other approaches for hierarchical process mining. Section 3 describes the details of the unsupervised hierarchical process mining approach PFM. In Sect. 4, we evaluate the performance of the overall approach and the different ranking approaches based on benchmarks known from the literature. Section 5 discusses the results, and Sect. 6 concludes the paper.

2 Related Work

Discovering process hierarchies from an event log requires the abstraction of events to higher-level events or the clustering of tasks into subprocesses. [16] presents a literature review on event abstraction in process mining. The authors present a taxonomy containing multiple dimensions to analyze different event abstraction methods. With regard to this work, the most important dimension is the supervision strategy, which describes whether the abstraction is done in a supervised or unsupervised manner. Supervised approaches require additional information on a hierarchy between events. According to [16], most approaches use some form of supervision. Meanwhile, the PFM approach proposed in this work is fully unsupervised.

The FlexHMiner presented in [10] uses the notion of *activity trees* to discover process hierarchy within an event log. The authors propose a fully supervised approach based on the task labels, as well as an unsupervised approach. The unsupervised approach uses random clustering to identify hierarchies. Measured by *fitness*, *precision*, *F1-Score*, *complexity*, and *size*, the authors find that the supervised approach outperforms the unsupervised approach, and both approaches outperform a flat process mining approach on said measures.

Besides supervised approaches based on task labels [9,10] proposes the use of multi-level event information to identify the hierarchy, called Multi-Level Miner (MLM). For MLM, multi-layer classifiers are chosen based on domain knowledge as well as by the analysis of short loops for attributes. The authors argue that the discovered process models are easier to understand than flat process models for users. In comparison, we aim for a purely unsupervised methodology that is applied to an event log before user interaction. The goal is to provide an understandable process model to the user, which can then be further refined by a domain expert.

Local Process Model (LPM) Discovery is described in [13,14] as the only approach to mine local process models with formal semantics. For each subprocess fragment, a process tree is created, which holds information on formal semantics. PFM adds the semantics to each subprocess fragment after having built a full process model from the subprocesses. To identify these process models, any mining approach can be used.

3 Methodology

The PFM approach uses an event log L as input. L comprises multiple events, each of which is associated with a unique case identifier of the associated activity and belongs to one process instance. A timestamp indicates when the event occurred, and a label identifies an event, formally: Let $\mathcal{A}$ be the set of all process activities, and TS be the set of all timestamps. The set of events E:$\subseteq \mathcal{A} \times \mathcal{TS}$ describes all events e=(a,ts) that reflect the execution of activity $a \in A$ at time $ts \in TS$. A trace $t :-< e_1, ..., e_n >$ is defined as a sequence of events that describe the execution of a sequence of activities for an underlying process. An event log L is a multiset of traces, $\mathcal{L}$ the set of all logs, $A_L : \mathcal{L} \mapsto \mathcal{A}$ determines the set of activities for which corresponding events are present in L.

PFM follows a structured, sequential, multi-stage processing pipeline. With the event log as input, this pipeline consists of the steps:

Step 1: Calculate the dependency matrix
Step 2: Calculate possible subprocess fragments
Step 3: Rank all fragments using one of three methods: Heuristic-Ranking (HR), Bigram-Ranking (BR), and Similarity-Ranking (SR).
Step 4: Find a disjoint set of fragments to build the full process model
Step 5: Discover the root process model and fragment process models

3.1 Step 1: Calculate Dependency Matrix with HeuristicsMiner

We calculate a dependency matrix M as it is used by the HeuristicsMiner [15], i.e. for log L, i.e., a quadratic matrix with row and columns determined by the set of activities A_L (see Table 1). The entries m_{ij} are determined based on the *strength of dependency* $Dep(a_i, a_j)$ between activities a_i and a_j which is calculated following the HeuristicsMiner formula:

$$\text{Dep}(a_i, a_j) = \frac{|a_i \rightarrow a_j| - |a_j \rightarrow a_i|}{|a_i \rightarrow a_j| + |a_j \rightarrow a_i| + 1} \quad \text{with} \tag{1}$$

$|a_i \rightarrow a_j| := |\{t \in L \mid \exists e = (a_i, ts_{a_i}), e' = (a_j, ts_{a_j}) \in t \wedge e \text{ directly followed by } e'\}|$

This formula yields a value in a range of $[-1, 1]$ that describes the sequential relationship between activity a_i and activity a_j. Following this, values close to 1 indicate a strong dependency from a_i towards a_j, i.e., a_i almost always leads to a_j. Values close to -1 indicate a strong inverse relationship, i.e., a_j more often leads to a_i. Values close to 0 do not indicate a clear dependency between activities (activities may occur independently or in parallel). Table 1 shows an exemplary dependency matrix derived from the event log. Each row holds the dependencies between a_i (row identifier) to all dependent tasks a_j (column identifier).

Table 1. Example event log (left) and its corresponding dependency matrix (right), mined using the HeuristicsMiner.

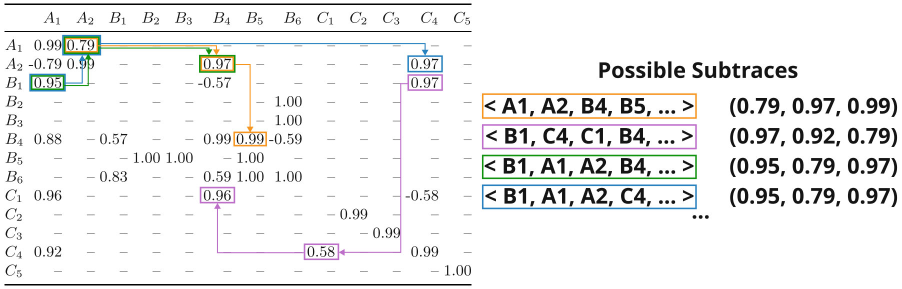
instance	event	timestamp
7ad	C_4	16:03:01
7ad	C_1	16:04:07
...	...	...
04c	A_2	16:04:52
04c	A_1	16:04:58
04c	A_1	16:05:14
...	...	...
c80	B_5	15:48:59
c80	B_5	15:49:10
c80	B_4	15:50:00

	A_1	A_2	B_1	B_2	B_3	B_4	B_5	B_6	C_1	C_2	C_3	C_4	C_5
A_1	0.99	0.79	–	–	–	–	–	–	–	–	–	–	–
A_2	-0.79	0.99	–	–	–	0.97	–	–	–	–	–	0.97	–
B_1	0.95	–	–	–	–	-0.57	–	–	–	–	–	0.97	–
B_2	–	–	–	–	–	–	–	1.00	–	–	–	–	–
B_3	–	–	–	–	–	–	–	1.00	–	–	–	–	–
B_4	0.88	–	0.57	–	–	0.99	0.99	-0.59	–	–	–	–	–
B_5	–	–	–	1.00	1.00	–	1.00	–	–	–	–	–	–
B_6	–	–	0.83	–	–	0.59	1.00	1.00	–	–	–	–	–
C_1	0.96	–	–	–	–	0.96	–	–	–	–	–	-0.58	–
C_2	–	–	–	–	–	–	–	–	–	0.99	–	–	–
C_3	–	–	–	–	–	–	–	–	–	–	0.99	–	–
C_4	0.92	–	–	–	–	–	–	–	0.58	–	–	0.99	–
C_5	–	–	–	–	–	–	–	–	–	–	–	–	1.00

Intuitively, to find traces in the matrix, a threshold could be defined that sets an acceptance level for a relationship between two tasks. Without a threshold, process traces can be found with an *all-activities-connected heuristic*, which always chooses the best candidate from the possible related candidates [15].

3.2 Step 2: Calculate Possible Subtraces

In contrast to the *all-activities-connected heuristic* [15], PFM finds a subtrace t^* by performing a depth-first search starting from every activity $a \in M$. A subtrace is represented as an ordered list of activities. Figure 1 shows how the traces are derived from the matrix. The universe of subtraces contains every possible subtrace starting from each activity in M. A subtrace t^* is not necessarily a subset of a trace $t \in L$. Since it is created from the dependency matrix M, additional subtraces are possible.



	A_1	A_2	B_1	B_2	B_3	B_4	B_5	B_6	C_1	C_2	C_3	C_4	C_5
A_1	0.99	0.79	–	–	–	–	–	–	–	–	–	–	–
A_2	-0.79	0.99	–	–	–	0.97	–	–	–	–	–	0.97	–
B_1	0.95	–	–	–	–	-0.57	–	–	–	–	–	0.97	–
B_2	–	–	–	–	–	–	–	1.00	–	–	–	–	–
B_3	–	–	–	–	–	–	–	1.00	–	–	–	–	–
B_4	0.88	–	0.57	–	–	0.99	0.99	-0.59	–	–	–	–	–
B_5	–	–	–	1.00	1.00	–	1.00	–	–	–	–	–	–
B_6	–	–	0.83	–	–	0.59	1.00	1.00	–	–	–	–	–
C_1	0.96	–	–	–	–	0.96	–	–	–	–	–	-0.58	–
C_2	–	–	–	–	–	–	–	–	–	0.99	–	–	–
C_3	–	–	–	–	–	–	–	–	–	–	0.99	–	–
C_4	0.92	–	–	–	–	–	–	–	0.58	–	–	0.99	–
C_5	–	–	–	–	–	–	–	–	–	–	–	–	1.00

Fig. 1. Deriving possible subtraces from a dependency matrix M starting at dependency (a_1, a_2).

Algorithm 1 describes how all subtraces are created from the dependency matrix M. It performs a depth-first search over the matrix and returns all possible subtraces by exploring all valid activity sequences. By setting a `threshold`, it can be specified which next activities are accepted to be added to a subtrace. The default for this `threshold` is 0. By setting `min_depth` and `max_depth`, a minimum and maximum size for subtraces can be set. In an unconstrained setting, finding all possible subtraces can lead to high computational complexity. Specifying minimum and maximum depth parameters for a subtrace or a high threshold can be used to constrain this step.

A found subtrace represents one candidate of a subprocess to hierarchically structure the process model of the event log. Resulting from the full set of subtraces, the goal is to find a combination of n subtraces that represent the root process in the best way with respect to some quality measure (see detailed explanation in Sect. 3.4).

3.3 Step 3: Ranking of Subtraces

Different approaches to measure the quality of a subtrace can be used to rank the identified subtraces. The quality of a subtrace describes how unambiguous the dependencies within a subtrace are. The ranking of subtraces is necessary for the recombination of the subtraces to the full process in the following step (see Sect. 3.4). We present three approaches that use the given dependency matrix M and the set of found subtraces. The three approaches span from a heuristic ranking approach, over the bigram ranking with interpretable co-occurrence features, to the Word2Vec-based ranking, which

Algorithm 1. Calculate all subtraces from the dependency matrix.

```
Require: dependencies (dictionary), threshold, max_depth, min_depth
Ensure: all_subtraces (list of traces)
all_subtraces ← empty list
procedure DFS(trace, depth)
    current ← last element of trace
    if depth ≥ max_depth then
        Append copy of trace to all_subtraces
        return
    end if
    next_nodes ← dependencies[current] (default empty dictionary)
    extended ← false
    for all (next_act, strength) in next_nodes do
        if strength ≥ threshold and next_act ∉ trace then
            Append next_act to trace
            DFS(trace, depth +1)
            Remove last element from trace
            extended ← true
        end if
    end for
    if not extended and depth ≥ min_depth then
        Append copy of trace to all_subtraces
    end if
end procedure
for all start_node in dependencies do
    DFS([start_node], 1)
end for
return all_subtraces
```

behaves more like a ‘black box’ due to its opaque vector representations and lack of interpretable features. Each of the ranking approaches creates a ranked list of possible subtraces as output.

Heuristic Ranking (HR). Given the dependency matrix M (see Eq. (1)), where m_{ij} represents the strength of a dependency from activity a_i to activity a_j, the heuristic ranking for a subtrace $t^* =< a_1, \dots, a_n >$ is defined as the product of the dependency strengths *Dep* along the trace t^*:

$$\mathrm{HR}(t^*) = \prod_{l=1}^{k-1} Dep_{a_l a_{l+1}} \tag{2}$$

This ranking approach uses the dependency matrix to assign higher rankings to variants with stronger cumulative dependencies. It is computationally inexpensive to calculate.

Bigram Ranking (BR). Bigram Ranking evaluates the likelihood of a trace based on bigram probabilities derived from the event log. A bigram matrix is constructed by counting transitions between pairs of activities in the log. To account for data sparsity and unseen transitions, smoothing is applied to the matrix M. We define the smoothed bigram probability of transitioning from activity a_i to activity a_j as:

$$\mathrm{P(a_j \mid a_i)} = \frac{C(a_i, a_j) + \alpha}{C(a_i) + \alpha \cdot |A|} \tag{3}$$

where:

- $C(a_i, a_j)$ is the count of transitions from a_i to a_j in the log,
- $C(a_i) = \sum_{j=1}^{k} C(a_i, a)$ is the total number of times a_i is followed by any activity,
- $|A|$ is the number of unique activities,
- α is a smoothing parameter (e.g., $\alpha = 1$ for Laplace smoothing).

The resulting *bigram probability matrix* $P \in \mathbb{R}^{k \times k}$ is defined as:

$$P = \begin{bmatrix} P(a_1 \mid a_1) & P(a_2 \mid a_1) & \cdots & P(a_k \mid a_1) \\ P(a_1 \mid a_2) & P(a_2 \mid a_2) & \cdots & P(a_k \mid a_2) \\ \vdots & \vdots & \ddots & \vdots \\ P(a_1 \mid a_k) & P(a_2 \mid a_k) & \cdots & P(a_k \mid a_k) \end{bmatrix}$$

Each row of the matrix corresponds to a source activity a_i, and each column corresponds to a target activity a_j. The matrix is row-normalized and reflects the smoothed probabilities of transitions observed in the event log.

The overall likelihood of a trace $t^* =< a_1, a_2, \ldots, a_n >$ is computed as the product of the *unigram probability* of the initial activity and the sequence of bigram transition probabilities between successive activities:

$$\mathrm{P}(t^*) = P(a_1) \cdot \prod_{i=2}^{n} P(a_i \mid a_{i-1}) \tag{4}$$

Here, $P(a_1)$ denotes the *unigram probability* of starting with activity a_1, while each $P(a_i \mid a_{i-1})$ represents a *bigram probability*, i.e., the conditional probability of activity a_i given the preceding activity a_{i-1}. These bigram probabilities are obtained from the *bigram probability matrix*.

Traces are then ranked based on their computed likelihood $P(t^*)$, with higher-probability traces being ranked higher.

Similarity Ranking (SR). As proposed in [8], we capture semantic similarity with a model trained on event log L. We leverage the functionality of the Python package Word2Vec to score the similarity of every task, which we then interpret as our ranking. Word2Vec directly computes a similarity score based on semantic similarity. Word2Vec is a neural embedding model that learns dense vector representations for words based on their distributional context. Similarity is measured by the cosine similarity in the embedding space. In accordance with [8], we embed traces as sentences and events as words. With the derived model from the event log, we compute the cosine similarity of an activity and its successor activity. Once the scores for all $n - 1$ consecutive activity pairs in a subtrace of length n have been computed, their mean is used as the ranking score for the subtrace. This process is repeated for each subtrace.

3.4 Step 4: Selection of a Disjoint Fragment Set Covering All Events

After identifying and ranking the subtraces, a strategy must be found to combine corresponding process fragments into a new process model. Let t^* be a subtrace. Then the corresponding process fragment $f_t := \{a \in \mathcal{A} \mid \exists\ e = (a, t_a) \in t\}$ is defined as the set of all activities with corresponding events in t^*. The problem of finding such a disjoint set corresponds to an instance of the Maximum-Weight Independent Set (MWIS) problem, which is known to be NP-hard [6]. In some special cases, the MWIS problem can be solved using a dynamic programming approach similar to the Knapsack problem [7]. This reduces the storage requirement from $\mathcal{O}\,(b \times n^*)$ to $\mathcal{O}(b)$ [5]. While optimizing storage constraints, runtime-wise, the completeness of the NP-hard problem makes it impossible to solve it optimally in a reasonable time for lots of possible combinations, e.g., a large number of small subtraces. To enable calculation within a reasonable runtime even for larger instances, we implemented a beam search strategy that reduces computational complexity by limiting the number of partial solutions explored at each step. This allows for a broader search space than pure greedy selection, while still maintaining practical efficiency. In addition, we further elaborated on the greedy approach to make it more comprehensive and interpretable, for instance, by explicitly enforcing disjointness and guaranteeing full event coverage. Together, these enhancements strike a balance between solution quality and runtime scalability.

Algorithm 2 describes a greedy approach to identify a disjoint set of fragments that collectively cover all events in the log. First, the best following trace is selected until no trace can be added further. If any events remain uncovered, meaning they are not part of any fragment and no suitable fragment can represent them, they are grouped into a final fragment to ensure complete log coverage.

Algorithm 2 iterates over the list of `ranked_fragments`, which is sorted in descending order based on the ranking described in Sect. 3.2. For each candidate fragment, the algorithm checks whether it overlaps with any previously selected fragment by performing a bitwise intersection between its mask and the accumulated mask of already used events. If the candidate does not share any events with the selected fragments, it is added to the result. After all candidates have been considered, the algorithm identifies any remaining uncovered events and groups them into a final fragment. This guarantees that the returned fragments are non-overlapping and collectively cover all events in the log.

The greedy approach offers favorable computational and memory complexity, as it processes each fragment in a single pass while maintaining a minimal state. However, further improvements have been proposed within the implementation of PFM to enhance the quality of the resulting scores while still ensuring reasonable runtime. These include techniques such as beam search and dynamic programming, which explore a broader space of candidate fragment combinations and can yield better overall solutions at the cost of increased, but still manageable, computational effort.

3.5 Step 5: Discover Root Process Model and Fragments Process Models

After identifying a final set of fragments, process models for each fragment and a full process can be mined. The top-level process model created from the identified frag-

Algorithm 2. Greedy Candidate State from Ranked Fragments.

Require:
`ranked_fragments`: list of $(score, trace, trace\_mask)$ sorted descending
`reconstruct_trace(mask)` ▷ build a trace fragment from a given mask

Ensure: A disjoint set of fragments whose union covers all events

$used\_mask \leftarrow 0$
$trace\_list \leftarrow [\,]$
$score\_list \leftarrow [\,]$
$total\_score \leftarrow 0$
for all $(score, trace, trace\_mask) \in ranked\_fragments$ **do**
  if $used\_mask \,\&\, trace\_mask \neq 0$ **then**
    continue ▷ Skip overlapping
  end if
  $used\_mask \leftarrow used\_mask \,|\, trace\_mask$
  append $trace$ **to** $trace\_list$
  append $score$ **to** $score\_list$
  $total\_score \leftarrow total\_score + score$
end for
▷ Add leftover events to ensure full coverage
$all\_mask \leftarrow$ mask of all events
if $used\_mask \neq all\_mask$ **then**
  $leftover\_mask \leftarrow all\_mask \,\&\, \sim used\_mask$
  $leftover\_trace \leftarrow$ `reconstruct_trace`$(leftover\_mask)$
  append $leftover\_trace$ **to** $trace\_list$ ▷ No change to total_score (no scoring for leftover)
end if
return $(used\_mask, total\_score, trace\_list, score\_list)$

ments is called the *root* model. The *root* model represents an abstracted version of the process itself. It shows the interaction of the found fragments for the given event log.

Within each of the found fragments, the process must also be mined. We call these models the *subprocess* models. A subprocess represents traces equivalent to the events in a fragment f in the event log L. Any miner could be used for this step. For PFM, we use the inductive miner [15] and the split miner [1] to mine both the *root* model and *subprocess* models. According to [2], the inductive miner and the split miner outperform other miners in quality and execution time.

Mining the Subprocess Models. To mine the *subprocess* models, the event log L is reduced for each fragment identified in Step 4 by a projection function $sp(L)$. The subprocess event log L_{sp} is created by keeping the events that correspond to an activity that is part of the selected fragment f. For each fragment in the set of selected fragments, L_{sp} is generated. This follows the projection function specified in [10]. The *subprocess* model can then be mined from this subprocess event log L_{sp} by applying a discovery algorithm.

Mining the Root Model. To generate the *root* model, an abstraction of the found fragments is needed. To do so, we use the abstraction function $f \uparrow$ described in [10] to abstract the log. For each trace in L, a single start event x^s and a single end event x^e are identified for each subprocess trace. Since we allow for fragments of size one, it is possible that a subprocess consists of one or multiple events for the same activity, i.e., a loop. If a subprocess consists of one single event, the timestamps for x^s and x^e are the same. If a subprocess consists of more than one event, i.e., a loop, x^s has the timestamp of the first occurrence, and x^e has the timestamp of the last occurrence. The resulting abstracted log $f \uparrow (L)$ consists only of these events and is then again mined with a discovery algorithm.

4 Evaluation

To evaluate PFM, we have prototypically implemented the approach (see Sect. 4.1). Moreover, we conduct several experiments (see Sect. 4.2–4.5) where we follow the quality measures proposed in [2] and compare the PFM approach with the findings of [10]. To run the experiments, we use the *cpd_benchmark* repository,[1] which implements the benchmarks proposed in [2].

4.1 Implementation of the Approach

PFM is implemented in python and is publicly available.[2] PFM uses the process discovery algorithms provided by ProM accessed through cpd_benchmark. To calculate the dependency matrix, PM4Py is used [4]. Creating and ranking all possible fragments for an event log is computationally highly expensive. For event logs with a large number of events, this calculation becomes intractable on standard machines. By setting reasonable values for the `maximum_depth` parameter and `threshold`, this calculation can be constrained in order to ensure reasonable computing times. Since the maximum depth parameter directly constrains the size that a fragment can take, the quality of the resulting hierarchical process models is affected by setting a low value.

The computationally hardest part is the combination of disjoint sets to rebuild the full process. It is an instance of the maximum-weight independent set problem and, therefore, NP-hard. In the implementation, finding a good solution to the problem is approached by allowing for two approaches to solve this. For smaller datasets, dynamic programming can be used. This approach finds qualitatively better solutions but is not recommended for event logs that lead to a large dependency matrix. In this case, beam search should be used.

4.2 Experimental Setups

For the evaluation, we compare the performance of the PFM approach and the different Ranking methods. Since the initial ranking of fragments has a big impact on the resulting hierarchical process model, the different ranking methods lead to process models of different quality. We evaluate the quality of each ranking method, namely Heuristic

[1] https://gitlab.com/janikbenzin/cpd_benchmark.
[2] https://github.com/jhtobis/process-fragment-miner.

Ranking (HR), Bigram Ranking (BR) and Similarity Ranking (SR) in two ways. Once for the resulting root model (Root) and once as proposed in [10] by averaging over the found fragments. We compare the quality scores with the reported scores for DK-FH* (Domain Knowledge approach) and RC-FH* (random clustering approach) [10]. Since [10] does not report quality measures for the resulting root model, the DK-FH and RC-FH approaches have been re-implemented and re-measured to enable a comparison (see DK, RC in Table 2).

To get a selection of disjoint fragment sets (Step 4), the DP approach is used for the datasets BPIC12, BPIC17_f, BPIC15_1f, ChessPiece. For the other datasets (BPIC15_{2-5}), beam search is used, due to the high number of activities present in the event logs. As described in [10], we employ the Inductive Miner with 0.2 path filtering (IMf) and the Split Miner (SM) with standard settings.

4.3 Data Sets

As in [10], we evaluate the PFM approach using publicly available real-life event logs from the BPIC Challenges, including BPIC12, a filtered collection of BPIC15, and a filtered version of BPIC17. The BPIC12 and BPIC17 Datasets have 36 and 18 unique activities. Meanwhile, the BPIC15 Log is split into five sublogs, which comprise 70, 82, 62, 65, and 74 activities. The BPIC15 sets allow for a much higher number of possible subprocess fragment candidates. The logs used for this study are the filtered logs as used in [2,10]. In addition to these logs, we add one event log "ChessPiece" which was produced with the process engine CPEE [11] and taken from a production use case. For this event log the hierarchical order was also known, to enable the use of DK-FH

4.4 Model Quality Measures

In accordance with [10], we use the five quality measures *fitness*, *precision*, *F1-Score*, *complexity*, and *size* to quantify the quality of the found process models. These measures have been proposed in [2]. To measure the quality of the root model, the abstraction function $f \uparrow$ is applied to the resulting event log, and the quality measures are calculated on the resulting event log. For the subprocess models, for each subprocess event log, the measures are calculated and then averaged.

The quality metrics of the derived subprocess models must also be interpreted in conjunction with those of the corresponding root models, as their performance and significance are contextually interdependent.

4.5 Results of Empirical Study

The findings of the experimental study are shown in Table 2. We compare our approach with the two approaches described in [10], namely RC as an unsupervised approach and DK as a supervised Domain Knowledge approach. DK and RC represent the results we achieved through a re-implementation of both approaches; RC-FH* and DK-FH* represent the results as reported in [10]. Instead of relying on a single execution as done in [10] for RC, we use the mean of 10 executions. The reason for this is the goal of better comparability by smoothing outliers, since RC is a fully randomized approach. For RC-FH*, the authors report a maximum fragment size of 10. The row labeled *IMF* reports the scores using the inductive miner [15], *SM* for the split miner [1]. *HR* describes Heuristic Ranking method, *BR* the Bigram Ranking method, and *SR* the Similarity ranking method.

Inductive and Split Miner. In combination with PFM, the IMF generally performs better than SM for the *fitness* metric. Conversely, the *precision* measure is higher for SM. The same accounts for the *F1-score*. For both root and mean metrics, the *F1-score* is higher when applying SM.

In terms of model size and complexity, SM generates smaller and less complex process models for both the root and the subprocess models. This effect is stronger for PMF than for the random clustering.

Comparison of Subprocess Means. When looking at the mean metrics over the subprocesses, the PFM approach achieves better results than RC when using SM. For BPIC12, PFM with SM matches or outperforms RC-FH* in all measures and RC in all measures but for *precision* with BR. With IMF for BPIC12, the HR and SR perform better than RC (RC-FH*). For the BPIC15_{1-5}f PFM achieves similar outcomes as RC, and in many cases achieves a better score than RC & RC-FH*.

Interestingly, for the "ChessPiece" event log generated with a process engine, PFM outperforms RC consistently. Also, the scores for the DK approach can be reached. For all other approaches, the supervised Domain Knowledge approach remains the best possible solution in terms of the quality measures.

Comparison of Root Model. When looking at the root model, PFM generally matches or improves the scores of RC, especially when using SM. For all BPIC15 sets, RC performs badly for precision when using IMF. The PFM approaches are more stable in that regard, except for SR for the Dataset BPIC15_3f. While RC underperforms for the

Table 2. Evaluation results.

			Root Metrics					Mean Metrics					
Log	**Alg**	**DAlg**	*Fi*	*Pr*	*F1*	*CFC*	*Size*	$\overline{Fi}$	$\overline{Pr}$	$\overline{F1}$	$\overline{CFC}$	$\overline{Size}$	*#SPs*
BPIC12	HR		**1.00**	0.64	0.78	10.00	24.00	**0.97**	**0.82**	**0.88**	8.00	15.00	5
	BR		**1.00**	0.75	0.86	10.00	27.00	0.94	0.78	0.84	7.83	14.17	6
	SR	**IMf**	0.99	0.65	0.79	11.00	24.00	0.96	**0.82**	**0.88**	**7.20**	**13.80**	5
	RC-FH*		–	–	–	–	–	**0.97**	0.78	0.86	20.00	35.00	4
	RC		0.99	**0.81**	**0.89**	**4.00**	**13.60**	0.96	0.73	0.82	15.67	25.53	3
	DK-FH*		–	–	–	–	–	0.96	0.78	0.86	20.00	36.00	4
	DK		1.00	0.82	0.90	8.00	18.00	0.98	0.76	0.85	16.33	26.67	–
	HR		0.94	0.87	0.90	10.00	22.00	0.92	**0.97**	0.94	8.00	14.40	5
	BR		0.94	0.88	0.91	16.00	30.00	0.94	0.93	0.93	**5.67**	**11.33**	6
	SR	**SM**	**0.95**	**1.00**	**0.98**	4.00	16.00	**0.95**	0.96	**0.95**	8.80	15.00	5
	RC-FH*		–	–	–	–	–	0.92	0.90	0.90	14.00	28.00	3
	RC		0.86	0.98	0.92	**3.50**	**11.60**	0.92	0.94	0.93	14.47	23.87	3
	DK-FH*		–	–	–	–	–	0.89	0.94	0.91	10.00	22.00	4
	DK		0.88	1.00	0.94	4.00	12.00	0.87	0.95	0.91	11.33	21.00	–

(*continued*)

Table 2. (*continued*)

Log	Alg	DAlg	*Fi*	*Pr*	*F1*	*CFC*	*Size*	$\overline{Fi}$	$\overline{Pr}$	$\overline{F1}$	$\overline{CFC}$	$\overline{Size}$	*#SPs*
BPIC15_1f	HR		0.95	**0.87**	**0.91**	**10.00**	**23.00**	**0.99**	0.77	0.86	20.60	34.60	5
	BR		0.95	**0.87**	**0.91**	**10.00**	**23.00**	**0.99**	0.77	0.86	20.60	34.60	5
	SR	**IMf**	0.97	0.80	0.87	13.00	29.00	0.98	0.77	0.86	20.00	34.00	5
	RC-FH*		–	–	–	–	–	0.97	0.84	0.88	16.00	29.00	9
	RC		**0.99**	0.35	0.52	18.30	38.00	0.96	**0.88**	**0.92**	**13.56**	**23.48**	8
	DK-FH*		–	–	–	–	–	**0.99**	0.87	0.91	9.00	19.00	15
	DK		0.99	0.73	0.84	18.00	38.00	0.99	0.92	0.95	11.75	20.12	–
	HR		0.78	**1.00**	0.88	**2.00**	**16.00**	**0.91**	0.98	**0.94**	9.60	24.80	5
	BR		0.78	**1.00**	0.88	**2.00**	**16.00**	**0.91**	0.98	**0.94**	9.60	24.80	5
	SR	**SM**	**0.89**	**1.00**	**0.94**	6.00	20.00	0.89	**0.99**	0.93	9.40	24.80	5
	RC-FH*		–	–	–	–	–	0.88	**0.99**	0.93	**8.00**	22.00	9
	RC		0.75	0.94	0.84	14.80	35.20	0.86	0.98	0.92	8.86	**19.36**	8
	DK-FH*		–	–	–	–	–	0.96	0.98	0.97	4.00	14.00	15
	DK		0.91	1.00	0.95	8.00	28.00	0.98	0.99	0.98	5.25	15.75	–
BPIC15_2f	HR		**0.98**	0.55	0.70	23.00	45.00	**0.97**	0.78	0.86	16.00	27.00	8
	BR		0.86	**0.81**	**0.83**	**9.00**	**32.00**	**0.97**	0.80	0.87	15.38	26.12	8
	SR	**IMf**	**0.98**	0.63	0.77	22.00	39.00	**0.97**	0.75	0.84	23.00	36.50	6
	RC-FH*		–	–	–	–	–	0.95	0.85	0.89	16.00	33.00	10
	RC		**0.98**	0.28	0.44	19.20	42.60	0.94	**0.89**	**0.92**	**14.34**	**25.31**	9
	DK-FH*		–	–	–	–	–	**0.97**	0.91	0.93	7.00	16.00	21
	DK		0.98	0.77	0.87	29.00	58.00	0.99	0.96	0.97	7.92	15.38	–
	HR		0.80	**1.00**	0.89	**2.00**	22.00	0.84	**0.98**	0.90	7.00	19.00	8
	BR		0.79	**1.00**	0.88	**2.00**	22.00	0.85	**0.98**	0.91	**5.75**	**17.75**	8
	SR	**SM**	**0.83**	**1.00**	**0.91**	**2.00**	**18.00**	**0.86**	**0.98**	**0.92**	9.67	24.67	6
	RC-FH*		–	–	–	–	–	0.83	0.96	0.89	11.00	26.00	10
	RC		0.70	0.97	0.81	10.80	33.40	0.80	0.97	0.87	9.28	20.21	9
	DK-FH*		–	–	–	–	–	0.94	0.99	0.97	4.00	13.00	21
	DK		0.72	0.97	0.83	12.00	42.00	0.97	0.99	0.98	3.69	11.77	–
BPIC15_3f	HR		0.97	0.66	0.78	18.00	40.00	0.95	0.83	0.88	13.29	**22.57**	7
	BR		0.93	**0.81**	**0.87**	6.00	**26.00**	0.97	0.83	0.89	13.57	22.71	7
	SR	**IMf**	**1.00**	0.47	0.63	29.00	52.00	**0.99**	0.76	0.86	14.71	23.29	7
	RC-FH*		–	–	–	–	–	0.94	0.87	**0.90**	16.00	33.00	8
	RC		0.98	0.59	0.73	**5.50**	**26.00**	0.93	**0.88**	**0.90**	**12.79**	23.63	7
	DK-FH*		–	–	–	–	–	0.97	0.94	0.95	6.00	15.00	17
	DK		0.99	0.64	0.78	22.00	43.00	0.99	0.96	0.97	8.80	15.90	–
	HR		0.85	**1.00**	0.92	**2.00**	**20.00**	**0.87**	**0.99**	**0.92**	**5.71**	**16.29**	7
	BR		**0.87**	**1.00**	**0.93**	4.00	22.00	0.85	0.98	0.91	6.57	17.14	7
	SR	**SM**	0.81	**1.00**	0.90	3.00	22.00	0.82	**0.99**	0.89	6.00	16.57	7
	RC-FH*		–	–	–	–	–	**0.87**	0.98	**0.92**	10.00	23.00	7
	RC		0.79	0.95	0.86	6.30	25.00	0.81	0.97	0.88	7.23	18.02	7
	DK-FH*		–	–	–	–	–	0.95	**0.99**	0.97	3.00	12.00	17
	DK		0.83	1.00	0.91	8.00	32.00	0.98	0.99	0.98	3.00	10.90	–
BPIC15_4f	HR		0.97	0.72	0.83	8.00	**26.00**	**0.97**	0.75	0.84	18.80	31.80	5
	BR		0.91	**0.88**	0.90	**5.00**	**26.00**	0.94	0.82	0.87	15.33	26.83	6
	SR	**IMf**	0.98	0.87	**0.92**	11.00	27.00	0.96	0.79	0.86	20.00	32.40	5
	RC-FH*		–	–	–	–	–	**0.97**	0.81	0.87	18.00	35.00	8
	RC		**0.99**	0.39	0.56	17.60	37.60	0.94	**0.86**	**0.90**	**14.44**	**25.43**	7
	DK-FH*		–	–	–	–	–	0.98	0.96	0.97	5.00	13.00	21
	DK		0.97	0.62	0.76	34.00	62.00	0.99	0.97	0.98	5.69	11.85	–
	HR		0.85	**1.00**	0.92	**0.00**	**14.00**	0.82	0.95	0.88	7.20	21.80	5
	BR		0.87	**1.00**	0.93	3.00	20.00	**0.83**	**0.99**	**0.90**	**5.17**	**17.83**	6
	SR	**SM**	**0.93**	0.99	**0.96**	5.00	20.00	**0.83**	0.97	0.89	7.20	21.80	5
	RC-FH*		–	–	–	–	–	**0.83**	0.98	**0.90**	8.00	24.00	8
	RC		0.80	0.98	0.88	6.10	24.30	0.80	0.97	0.87	7.06	18.38	7
	DK-FH*		–	–	–	–	–	0.95	**0.99**	0.97	2.00	11.00	21
	DK		0.72	0.98	0.83	8.00	38.00	0.96	0.99	0.98	2.00	8.85	–

(*continued*)

Table 2. (*continued*)

Log	Alg	DAlg	*Fi*	*Pr*	*F1*	*CFC*	*Size*	$\overline{Fi}$	$\overline{Pr}$	$\overline{F1}$	$\overline{CFC}$	$\overline{Size}$	*#SPs*
BPIC15_1f	HR		0.95	**0.87**	**0.91**	**10.00**	**23.00**	**0.99**	0.77	0.86	20.60	34.60	5
	BR		0.95	**0.87**	**0.91**	**10.00**	**23.00**	**0.99**	0.77	0.86	20.60	34.60	5
	SR	**IMf**	0.97	0.80	0.87	13.00	29.00	0.98	0.77	0.86	20.00	34.00	5
	RC-FH*		–	–	–	–	–	0.97	0.84	0.88	16.00	29.00	9
	RC		**0.99**	0.35	0.52	18.30	38.00	0.96	**0.88**	**0.92**	**13.56**	**23.48**	8
	DK-FH*		–	–	–	–	–	**0.99**	0.87	0.91	9.00	19.00	15
	DK		0.99	0.73	0.84	18.00	38.00	0.99	0.92	0.95	11.75	20.12	–
	HR		0.78	**1.00**	0.88	**2.00**	**16.00**	**0.91**	0.98	**0.94**	9.60	24.80	5
	BR		0.78	**1.00**	0.88	**2.00**	**16.00**	**0.91**	0.98	**0.94**	9.60	24.80	5
	SR	**SM**	**0.89**	**1.00**	**0.94**	6.00	20.00	0.89	**0.99**	0.93	9.40	24.80	5
	RC-FH*		–	–	–	–	–	0.88	**0.99**	0.93	**8.00**	22.00	9
	RC		0.75	0.94	0.84	14.80	35.20	0.86	0.98	0.92	8.86	**19.36**	8
	DK-FH*		–	–	–	–	–	0.96	0.98	0.97	4.00	14.00	15
	DK		0.91	1.00	0.95	8.00	28.00	0.98	0.99	0.98	5.25	15.75	–
BPIC15_2f	HR		**0.98**	0.55	0.70	23.00	45.00	**0.97**	0.78	0.86	16.00	27.00	8
	BR		0.86	**0.81**	**0.83**	**9.00**	**32.00**	**0.97**	0.80	0.87	15.38	26.12	8
	SR	**IMf**	**0.98**	0.63	0.77	22.00	39.00	**0.97**	0.75	0.84	23.00	36.50	6
	RC-FH*		–	–	–	–	–	0.95	0.85	0.89	16.00	33.00	10
	RC		**0.98**	0.28	0.44	19.20	42.60	0.94	**0.89**	**0.92**	**14.34**	**25.31**	9
	DK-FH*		–	–	–	–	–	**0.97**	0.91	0.93	7.00	16.00	21
	DK		0.98	0.77	0.87	29.00	58.00	0.99	0.96	0.97	7.92	15.38	–
	HR		0.80	**1.00**	0.89	**2.00**	22.00	0.84	**0.98**	0.90	7.00	19.00	8
	BR		0.79	**1.00**	0.88	**2.00**	22.00	0.85	**0.98**	0.91	**5.75**	**17.75**	8
	SR	**SM**	**0.83**	**1.00**	**0.91**	**2.00**	**18.00**	**0.86**	**0.98**	**0.92**	9.67	24.67	6
	RC-FH*		–	–	–	–	–	0.83	0.96	0.89	11.00	26.00	10
	RC		0.70	0.97	0.81	10.80	33.40	0.80	0.97	0.87	9.28	20.21	9
	DK-FH*		–	–	–	–	–	0.94	0.99	0.97	4.00	13.00	21
	DK		0.72	0.97	0.83	12.00	42.00	0.97	0.99	0.98	3.69	11.77	–
BPIC15_3f	HR		0.97	0.66	0.78	18.00	40.00	0.95	0.83	0.88	13.29	**22.57**	7
	BR		0.93	**0.81**	**0.87**	6.00	**26.00**	0.97	0.83	0.89	13.57	22.71	7
	SR	**IMf**	**1.00**	0.47	0.63	29.00	52.00	**0.99**	0.76	0.86	14.71	23.29	7
	RC-FH*		–	–	–	–	–	0.94	0.87	**0.90**	16.00	33.00	8
	RC		0.98	0.59	0.73	**5.50**	**26.00**	0.93	**0.88**	**0.90**	**12.79**	23.63	7
	DK-FH*		–	–	–	–	–	0.97	0.94	0.95	6.00	15.00	17
	DK		0.99	0.64	0.78	22.00	43.00	0.99	0.96	0.97	8.80	15.90	–
	HR		0.85	**1.00**	0.92	**2.00**	**20.00**	**0.87**	**0.99**	**0.92**	**5.71**	**16.29**	7
	BR		**0.87**	**1.00**	**0.93**	4.00	22.00	0.85	0.98	0.91	6.57	17.14	7
	SR	**SM**	0.81	**1.00**	0.90	3.00	22.00	0.82	**0.99**	0.89	6.00	16.57	7
	RC-FH*		–	–	–	–	–	**0.87**	0.98	**0.92**	10.00	23.00	7
	RC		0.79	0.95	0.86	6.30	25.00	0.81	0.97	0.88	7.23	18.02	7
	DK-FH*		–	–	–	–	–	0.95	**0.99**	0.97	3.00	12.00	17
	DK		0.83	1.00	0.91	8.00	32.00	0.98	0.99	0.98	3.00	10.90	–
BPIC15_4f	HR		0.97	0.72	0.83	8.00	**26.00**	**0.97**	0.75	0.84	18.80	31.80	5
	BR		0.91	**0.88**	0.90	**5.00**	**26.00**	0.94	0.82	0.87	15.33	26.83	6
	SR	**IMf**	0.98	0.87	**0.92**	11.00	27.00	0.96	0.79	0.86	20.00	32.40	5
	RC-FH*		–	–	–	–	–	**0.97**	0.81	0.87	18.00	35.00	8
	RC		**0.99**	0.39	0.56	17.60	37.60	0.94	**0.86**	**0.90**	**14.44**	**25.43**	7
	DK-FH*		–	–	–	–	–	0.98	0.96	0.97	5.00	13.00	21
	DK		0.97	0.62	0.76	34.00	62.00	0.99	0.97	0.98	5.69	11.85	–
	HR		0.85	**1.00**	0.92	**0.00**	**14.00**	0.82	0.95	0.88	7.20	21.80	5
	BR		0.87	**1.00**	0.93	3.00	20.00	**0.83**	**0.99**	**0.90**	**5.17**	**17.83**	6
	SR	**SM**	**0.93**	0.99	**0.96**	5.00	20.00	**0.83**	0.97	0.89	7.20	21.80	5
	RC-FH*		–	–	–	–	–	**0.83**	0.98	**0.90**	8.00	24.00	8
	RC		0.80	0.98	0.88	6.10	24.30	0.80	0.97	0.87	7.06	18.38	7
	DK-FH*		–	–	–	–	–	0.95	**0.99**	0.97	2.00	11.00	21
	DK		0.72	0.98	0.83	8.00	38.00	0.96	0.99	0.98	2.00	8.85	–

BPIC datasets, it performs very well on the ChessPiece dataset. For the Root Metrics, DK is often outperformed. Only in a few cases, such as for BPIC15_1f, DK strikes the best Fitness score alone.

Comparison of Different Ranking Models. Considering the different ranking methods, HR, BR, and SR, no clear-cut best approach can be identified. SR performs very well, yet has certain steep drops in precision, e.g., for the "ChessPiece" dataset. Among all datasets and both root and mean metric, the BR approach is the most stable and reaches the best values the most. It additionally does not underperform for any of the datasets. The risk of harsh outliers for BR is smaller than for the other two ranking approaches. Most importantly, the BR performs consistently well for both root and mean metrics on the same dataset.

Table 2 shows that the PFM approach can achieve good outcomes in mining hierarchical process models. Looking at the root and mean metrics together, we can see that the PFM approaches perform more uniformly for the two metrics than the random approach. Especially for the larger Datasets BPIC15_{1-5}f in combination with IMF, the RC approach drops in performance for *precision*. Meanwhile, the PFM approach with the BR ranking method and SM performs stable on both performance metrics and is the most consistent of the unsupervised approaches. While the supervised approach DK often performs best, in certain settings, e.g., for the "ChessPiece" dataset, it can be outperformed by the unsupervised PFM approach. DK also reaches comparably low scores for the root model on the BPIC15_{1-5}f datasets. Here, the PFM shows its strength for *precision* in combination with SM.

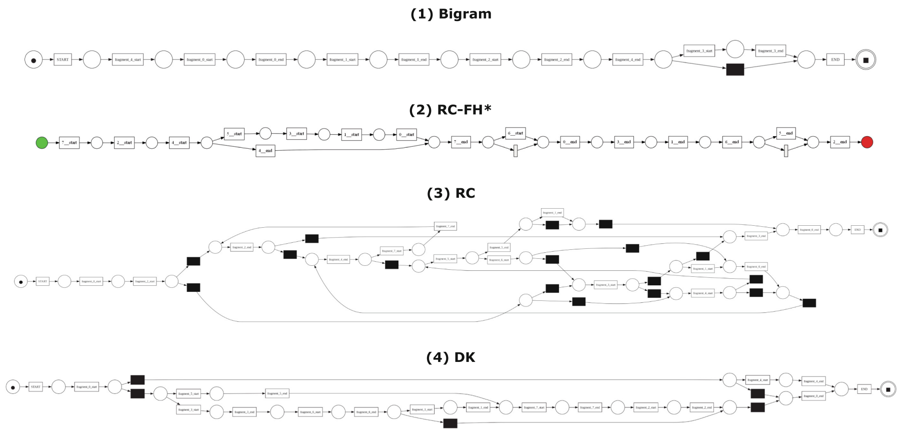


Fig. 2. Comparison of bigram (1), RC-FH* (2), RC (3), and DK (4) on BPIC15_1f data set with SM, where (2) is taken from [10].

5 Discussion

Abstracting flat event logs into more hierarchical ones brings positive effects in terms of understandability for a user. Yet, it is unclear how the simplification achieved through a hierarchical process model can be measured best, or how hierarchical process models could be compared to flat ones [9]. While the measures defined in [2] mark important quality benchmarks for process mining, it can be seen that they are not well suited for hierarchical process mining. As shown in [10], comparing flat approaches with a hierarchical modeling approach can lead to big improvements for these quality measures, even when using random clustering (RC). The measures *fitness*, *precision*, and *F1-score* disregard the semantics in the abstracted process models. While this is acceptable for flat process models mined on complete event logs, for subsets of an event log, this measure leads to confusion. The mean metric on subprocesses can give no good insight into the found process model since noisy events are discarded; therefore, we also measured the root metric in this work and consider root and mean metrics both separately, and interdependently. To this end, an approach to measure the simplicity and understandability of a hierarchical process model is both missing and needed. It cannot be quantified by the current quality measures whether the hierarchical process models found by any approach are better understood by domain experts than flat approaches. Figure 2 presents the found root models for BPIC15_1f with Bigram Ranking (1), RC-FH*(2), RC(3), and DK(4). We can see that PFM with the Bigram Ranking (1) leads to a clean, sequential process model, which allows for the exclusion of Fragment_3. Meanwhile, the random approaches RC-FH* and RC lead to process models in which the subprocesses are heavily interlinked and no structure can be identified. Yet, RC performs similarly well on the quality measures for this data set, as the other approaches (see Table 2). The DK approach and the Bigram ranking provide both good and well-structured root models.

Future work on hierarchical process models should focus on finding appropriate quality measures to compare the found models for both the root and the subprocess models in combination. These measures should also strive to consider the semantics of a process model.

6 Conclusion and Outlook

In this work, we presented PFM, an unsupervised hierarchical process mining approach based on the dependency matrix generated by the HeuristicsMiner [15]. At the core of the approach is the creation and ranking of subtraces from an event log. We developed and tested three different ranking methods for subtraces. We show that the identification of hierarchical models without any additional information, such as context data or labeling, is possible and can lead to comparable outcomes as supervised approaches. To do so, the PFM approach was evaluated against existing supervised and unsupervised approaches by applying known process mining quality metrics on the mined subprocess models and the identified root models. We find that PFM works best and most consistently over different datasets when using it in combination with the Split Miner and the Bigram Ranking method. The source code for PFM is available.

In future work, the PFM approach should be integrated into a software tool, which enables users and domain experts to iteratively analyse and define hierarchies in processes in an intertwined way. This will help bridge the typical process mining workflow from preprocessing to mining & analysis [3]. This work offers different directions to further improve the performance of PFM. Through a more intertwined approach of building subprocess fragments and ranking them, the computational overhead could be reduced by focusing on only computing higher-ranked fragments. Formulating the recombination problem as a combinatorial optimization problem is also an interesting research direction. With PFM, we present an unsupervised hierarchical process mining approach, which can be applied to any event log without the need for specific log pre-processing. It can be used to create a meaningful overview of a process at the beginning of a process mining and optimization project. PFM enables domain experts to quickly get a concise overview of a process.

Acknowledgments. This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – GRK2201 – Projektnummer - 277991500.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Augusto, A., Conforti, R., Dumas, M., La Rosa, M., Polyvyanyy, A.: Split miner: automated discovery of accurate and simple business process models from event logs. Knowl. Inf. Syst. **59**(2), 251–284 (2019)
2. Augusto, A., et al.: Automated discovery of process models from event logs: review and benchmark. IEEE Trans. Knowl. Data Eng. **31**(4), 686–705 (2019)
3. Beerepoot, I., et al.: The biggest business process management problems to solve before we die. Comput. Ind. **146**, 103837 (2023)
4. Berti, A., van Zelst, S., Schuster, D.: Pm4py: a process mining library for python. Softw. Impacts **17**, 100556 (2023)
5. Chebil, K., Khemakhem, M.: A dynamic programming algorithm for the knapsack problem with setup. Comput. Oper. Res. **64**, 40–50 (2015)
6. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Company (1979)
7. Kellerer, H., Pferschy, U., Pisinger, D.: Knapsack Problems. Springer (2004)
8. Koninck, P.D., vanden Broucke, S., Weerdt, J.D.: act2vec, trace2vec, log2vec, and model2vec: representation learning for business processes. In: Business Process Management, pp. 305–321 (2018)
9. Leemans, S.J., Goel, K., Van Zelst, S.J.: Using multi-level information in hierarchical process mining: balancing behavioural quality and model complexity. In: Process, pp. 137–144 (2020)
10. Lu, X., Gal, A., Reijers, H.A.: Discovering hierarchical processes using flexible activity trees for event abstraction. In: Process Mining, pp. 145–152 (2020)
11. Mangler, J., Rinderle-Ma, S.: CPEE-cloud process exection engine. In: Business Process Management Demos, pp. 51–55 (2014)
12. Mendling, J., Reijers, H.A., van der Aalst, W.M.P.: Seven process modeling guidelines (7PMG). Inf. Softw. Technol. **52**(2), 127–136 (2010)
13. Tax, N., Dalmas, B., Sidorova, N., van der Aalst, W.M.P., Norre, S.: Interest-driven discovery of local process models. Inf. Syst. **77**, 105–117 (2018)

14. Tax, N., Sidorova, N., Haakma, R., van der Aalst, W.M.P.: Mining local process models. J. Innov. Digit. Ecosyst. **3**(2), 183–196 (2016)
15. Weijters, A.J., van Der Aalst, W.M., De Medeiros, A.A.: Process mining with the HeuristicsMiner algorithm. Technische Universiteit Eindhoven (2006)
16. van Zelst, S.J., Mannhardt, F., de Leoni, M., Koschmider, A.: Event abstraction in process mining: literature review and taxonomy. Granular Comput. **6**(3), 719–736 (2021)