Heuristics for High-Utility Local Process Model Mining

Benjamin Dalmas[1], Niek Tax[2], and Sylvie Norre[1]

[1] Clermont-Auvergne University, LIMOS CNRS UMR 6158, Aubière, France
{benjamin.dalmas,sylvie.norre}@isima.fr
[2] Eindhoven University of Technology, Department of Mathematics and Computer Science, P.O. Box 513, 5600MB Eindhoven, The Netherlands
n.tax@tue.nl

Abstract. Local Process Models (LPMs) describe structured fragments of process behavior occurring in the context of less structured business processes. In contrast to traditional support-based LPM discovery, which aims to generate a collection of process models that describe highly frequent behavior, High-Utility Local Process Model (HU-LPM) discovery aims to generate a collection of process models that provide useful business insights by specifying a *utility function*. Mining LPMs is a computationally expensive task, because of the large search space of LPMs. In support-based LPM mining, the search space is constrained by making use of the property that support is anti-monotonic. We show that in general, we cannot assume a provided utility function to be anti-monotonic, therefore, the search space of HU-LPMs cannot be reduced without loss. We propose four heuristic methods to speed up the mining of HU-LPMs while still being able to discover useful HU-LPMs. We demonstrate their applicability on three real-life data sets.

Keywords: Process Discovery, Pattern Mining, Heuristics

1 Introduction

Process Mining [1] has emerged as a new discipline aiming at the improvement of business processes through the analysis of event data recorded by information systems. An event log contains recorded *events* related to a process execution. Events consist of a case identifier (grouping together events that belong to the same process instance), and information on what was performed, when, by whom, etc. Process discovery techniques aim to discover an interpretable model that accurately describes the process from such an event log. The process models obtained with process discovery give insight in what is happening in the process, and can be used as a starting point for different types of further analysis, e.g. bottleneck analysis [17], and comparison of the same process between organizations [6]. Many algorithms have been proposed for process discovery, e.g., [3, 4, 14–16].

Recently, Local Process Model (LPM) discovery [20, 23] has emerged, which is concerned with the discovery of a ranking of process models (i.e., LPMs),

where each individual LPM describes only a subset of the process activities. LPMs aim to describe frequent local pieces of behavior, therefore, LPMs can be seen as a special form of *frequent patterns* [10], where each pattern is a process model. However, in contrast to other pattern mining approaches that operate on sequence data, such as episode mining [13] and sequential pattern mining [19], LPMs are not limited to subsequences or partial orders and can additionally consist of more complex structures, such as loops and choices.

A recent trend in the frequent pattern mining field is to take the relative importance of the activities in the log into account in the knowledge discovery process. This results in the discovery of patterns that address business concerns, e.g. high financial costs, instead of the discovery of simply the most frequent patterns. In previous work we introduced high-utility local process models (HU-LPMs) [22] to bridge the concept of utility based discovery into the process mining field, and adapted it to the logging concepts typically seen in process mining event logs, such as event attributes, trace attributes, etc. In the pattern mining field, the concept of *utility* is often defined narrower and solely based on the set of activities that is described by a pattern.

To deal with the computational complexity of searching patterns with such a rich set of constructs that are supported by LPMs (i.e., sequential orderings, parallel blocks, loops, choices), a support-based pruning strategy [23] as well as a set of heuristic approaches [20] have been introduced for the discovery of LPMs. However, support-based pruning and the existing set of heuristic approaches for LPM discovery cannot be used for the discovery of high-utility LPM discovery, as LPMs with high utility can be infrequent. Furthermore, the utility of an LPM is not necessarily monotonic, i.e., an LPM that does not meet a utility threshold can still be expanded into an LPM that does meet this threshold.

In this paper we propose four different heuristic approaches to prune the search space of the HU-LPM discovery task. We perform experiments on three different logs and show that our approaches speed up the discovery of HU-LPMs, while still being able to discover useful HU-LPMs. The techniques described in this paper have been implemented in the ProM process mining framework [9] as part of the *LocalProcessModelDiscovery*[1] package.

This paper is organized as follows. Section 2 describes related work. Section 3 introduces the basic concepts used in this paper. In Section 4, we introduce the four heuristic approaches for HU-LPM mining. We discuss the experimental setup and experimental results in Section 5. Finally, we conclude and discuss future areas of research in Section 6.

2 Related Work

In the pattern mining discipline, the limitations of support-based mining have become apparent in recent years, and as a result the interest has grown in high-utility patterns; i.e., patterns providing useful business insight. This has led to an increasing number of methods and techniques that address the high-utility mining

[1] https://svn.win.tue.nl/repos/prom/Packages/LocalProcessModelDiscovery/

(HUM) problem [8, 24–26]. USpan uses a *lexicographic quantitative sequence tree (LSQ-tree)* to extract the complete set of high utility sequences [24]. A LQS-tree is a tree structure where each node stores a sequence of activities and its utility. The sequence stored by a node being a super-sequence of the sequence stored by the node's parent, this type of structure allows for fast access and updates when mining high-utility patterns. A similar tree structure, the *HUSP-Tree* is used by the HUSP-Stream algorithm to enable fast updates when mining high-utility patterns from sequential data streams [26]. The problem of mining incremental sequential datasets is also addressed in [8], using an efficient indexing strategy. In [25], the HUSRM algorithm efficiently mines sequential rules using a *utility table*, a novel data structure to support recursive rule expansions.

The utility in sequential patterns is regarded to be the sum of the utility of the activities that fit the sequential pattern. The majority of pruning strategies that are used in HUM algorithms are based on *Transaction-Weighted Utility* (TWU). The TWU of a pattern X is the sum of utilities of the sequences containing X, resulting in an upper bound for the utility of pattern X that can be computed efficiently. In case the TWU of a pattern X does not meet a predefined minimum threshold, X can be safely pruned since its actual utility can only be lower than or equal to TWU. In traditional HUM algorithms, the utility function is defined on the activity level; i.e., each activity in the dataset is given a utility and the utility of a pattern is the sum of all activity utilities. Therefore, TWU and other activity-based pruning strategies can be used for efficient pruning for HU-LPM mining when utility is defined on the activity level. However, utility functions of HU-LPMs are defined in a more general way, allowing utility for example to depend also on event attributes or trace attributes instead of the activity. Therefore, TWU cannot be used to prune the search space of HU-LPMs. With sequence-based pruning strategies being inapplicable in HU-LPM mining, we investigate in this paper utility-based heuristics.

3 Preliminaries

In this section we introduce notations related to event logs, Local Process Models (LPMs) and High-Utility Local Process Models (HU-LPMs) which are used in later sections of this paper.

3.1 Events, Traces, and Event Logs

X^* denotes the set of all sequences over a set X and $\sigma = \langle a_1, a_2, \ldots, a_n \rangle$ a sequence of length n, with $\sigma(i) = a_i$ and $|\sigma| = n$. $\langle\rangle$ is the empty sequence and $\sigma_1\sigma_2$ is the concatenation of sequences σ_1 and σ_2. We denote with $\sigma{\restriction_X}$ the projection of sequence σ on set X, e.g., for $\sigma = \langle a, b, c \rangle$, and $X = \{a, c\}$, $\sigma{\restriction_X} = \langle a, c \rangle$.

In the context of process logs, we assume the set of all *process activities* Σ to be given. An *event* e in an event log is the occurrence of an activity $e{\in}\Sigma$. We call a sequence of events $\sigma{\in}\Sigma^*$ a *trace*. An *event log* $L{\in}\mathbb{N}^{\Sigma^*}$ is a finite

multiset of traces. For example, the event log $L = [\langle a, b, c\rangle^2, \langle b, a, c\rangle^3]$ consists of 2 occurrences of trace $\langle a, b, c\rangle$ and three occurrences of trace $\langle b, a, c\rangle$. We lift projection of sequences to multisets of sequences, e.g., $L{\restriction}_{\{a,c\}} = [\langle a, c\rangle^5]$.

3.2 Local Process Models

LPMs [23] are process models that describe frequent but partial behavior; i.e., they model a subset of the activities of the process, seen in the event log. An iterative expansion procedure is used in [23] to generate a ranked collection of LPMs. LPMs are limited to 5 activities as the expansion procedure is a combinatorial problem of which the size depends on the number of activities in the event log as well as the maximum number of activities in the LPMs that are mined. Though LPMs can be represented in any process modeling notation, such as BPMN [18], UML [11], or EPC [12], here we use Process Trees [5] to represent LPMs.

A process tree is a tree structure where leaf nodes represent activities. The non-leaf nodes represent *operators*, which specify the allowed behavior over the activity nodes. Allowed operator nodes are the *sequence* operator ($\rightarrow$) that indicates that the first child is executed before the second, the *exclusive choice* operator ($\times$) that indicates that exactly one of the children can be executed, the *concurrency* operator ($\wedge$) that indicates that every child will be executed but allows for any ordering, and the *loop* operator ($\circlearrowleft$), which has one child node and allows for repeated execution of this node. $\mathfrak{L}(LPM)$ represents the language of process tree LPM, i.e., the set of sequences allowed by the model. Figure 1d shows an example process tree M_4, with $\mathfrak{L}(M_4)$=$\{\langle A, B, C\rangle, \langle A, C, B\rangle, \langle D, B, C\rangle, \langle D, C, B\rangle\}$. Informally, it indicates that either activity A or D is executed first, followed by the execution of activities B and C in any order.

A technique to generate a ranked collection of LPMs through iterative expansion of candidate process trees is proposed in [23]. The expansion procedure consists in the replacement of one of the leaf activity node a of the process tree by an operator node (i.e., $\rightarrow$,$\times$,$\wedge$, or $\circlearrowleft$), where one of the child nodes is the replaced activity node a and the other is a new activity node b. $\mathcal{M}$ is the LPM universe; i.e., the set of all possible LPMs. An LPM $M{\in}\mathcal{M}$ can be expanded in many ways, as it can be extended by replacing any one of its activity nodes, expanding it with any of the operator nodes, and with a new activity node that represents any of the activities present in the event log. We define $Exp(M)$ as the set of expansions of M, and *exp_max* the maximum number of expansions allowed from an *initial LPM*; i.e., an LPM containing only one activity.

Figure 1 illustrates the expansion procedure, starting from the initial LPM M_1 of Figure 1a. The LPM of Figure 1a is first expanded into a larger LPM by replacing A by operator node $\rightarrow$, with activity A as its left child node and B its right child node, resulting in the LPM of Figure 1b. Note that M_1 can also be expanded in other ways, and LPM discovery recursively explores all possible process trees that meet a support threshold by iterative expansion. In a second expansion step, activity node B of the LPM of Figure 1b is replaced by operator node $\wedge$, with activity B as its left child and C its right child, resulting in Figure

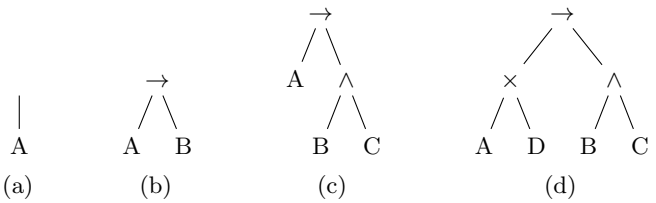


Fig. 1. *(a)* An initial LPM M_1 and *(b)* M_2, *(c)* M_3, *(d)* M_4, three LPM built from successive expansions.

event id	activity	time	cost
1	A	26-3-2017 13:00	€ 100
2	D	26-3-2017 13:27	€ 200
3	B	26-3-2017 13:25	€ 300
4	C	26-3-2017 13:35	€ 100
5	D	26-3-2017 13:42	€ 400
6	C	26-3-2017 15:47	€ 200
7	A	26-3-2017 16:10	€ 100
8	C	26-3-2017 16:34	€ 400
9	B	26-3-2017 16:52	€ 300
10	D	26-3-2017 16:59	€ 200
11	A	26-3-2017 17:13	€ 1000
12	B	26-3-2017 17:15	€ 1000
13	A	26-3-2017 17:16	€ 150

(a)

σ = ⟨A,D,B,C,D,C,A,C,B,D,A,B,A⟩
$\sigma\restriction_{\{A,B,C\}}$ = ⟨A,B,C,C,A,C,B,A,B,A⟩
λ_1 γ_1 λ_2 γ_2 λ_3
$\Gamma_{\sigma,LPM}$ = ⟨A,B,C,A,C,B⟩

(b)

Fig. 2. *(a)* A trace σ of an event log L. *(b)* The segmentation of σ on M_3.

1c. Finally, activity node A of the LPM of Figure 1c is replaced by operator node $\times$ with as left child activity A and as right child activity D, forming the LPM of Figure 1d. In traditional LPM discovery the expansion procedure of an LPM stops when the behavior described by the LPM is not observed frequently enough in an event log L (i.e., with regard to some *support threshold*).

To evaluate a given LPM on a given event log L, its traces $\sigma \in L$ are first projected on the set of activities X in the LPM, i.e., $\sigma' = \sigma\restriction_X$. The projected trace σ' is then segmented into γ-segments that fit the behavior of the LPM and λ-segments that do not fit the behavior of the LPM, i.e., $\sigma' = \lambda_1\gamma_1\lambda_2\gamma_2 \cdots \lambda_n\gamma_n\lambda_{n+1}$ such that $\gamma_i \in \mathfrak{L}(LPM)$ and $\lambda_i \notin \mathfrak{L}(LPM)$. We define $\Gamma_{\sigma,LPM}$ to be a function that projects trace σ on the LPM activities and obtains its subsequences that fit the LPM, i.e., $\Gamma_{\sigma,LPM} = \gamma_1\gamma_2 \ldots \gamma_n$.

Let our LPM M_3 under evaluation be the process tree of Figure 1c and let σ be the example trace shown in Figure 2a. Function $Act(LPM)$ obtains the set of process activities in the LPM, e.g. $Act(M_3) = \{A, B, C\}$. Projection on the activities of the LPM gives $\sigma\restriction_{Act(M_3)} = \langle A, B, C, C, A, C, B, A, B, A\rangle$. Figure 2b shows the segmentation of the projected trace on the LPM, leading to $\Gamma_{\sigma,LPM} = \langle A, B, C, A, C, B\rangle$. The segmentation starts with an empty non-fitting segment λ_1, followed by a fitting segment $\gamma_1 = \langle A, B, C\rangle$, which completes one run through the process tree. The second event C in σ cannot be replayed on LPM, since it only allows for one C and γ_1 already contains a C. This results in a non-fitting segment $\lambda_2 = \langle C\rangle$. $\gamma_2 = \langle A, C, B\rangle$ again represents a run through

process tree, the segmentation ends with non-fitting segment $\lambda_3=\langle A, B, A\rangle$. We lift segmentation function Γ to event logs, $\Gamma_{L,LPM}=\{\Gamma_{\sigma,LPM}|\sigma\in L\}$. An alignment-based [2] implementation of Γ, as well as a method to rank and select LPMs based on their support, i.e., the number of events in $\Gamma_{L,LPM}$, is described in [23].

Several metrics are taken into account to assess the quality of an LPM, but all of them are support-based and depend on the number of events in $\Gamma_{L,LPM}$. We refer the reader to [23] for detailed and formal definitions of LPMs, extensions of LPMs, and evaluating LPMs on logs.

Definition 1. ***Local Process Model Mining Problem:*** *Given an event log L, the LPM mining problem is defined as the task of discovering a set of frequent LPMs, where the total number of fragments replayable is above a defined threshold.*

3.3 High-Utility Local Process Models

A High-Utility Local Process Model (HU-LPM) is an LPM where (i) its importance is related to the *utility* of the fragments it can replay instead of the number of fragments it can replay and (ii) where this utility is above a predefined threshold. Note that HU-LPMs are a generalization of LPMs, as we have shown in [22] that the quality measures that are used in support-based LPM mining can be expressed as utility functions for HU-LPM mining. Several scopes on which utility functions can be defined are described in [22]:

Trace the most general class of utility functions, the trace-level utility functions allow the utility of fitting trace fragments to depend on the events in these specific fragments, their attributes and properties of the case itself. An example of trace-level utility function is the search for LPMs that explain a high share of the total running time of a case.

Event this class of utility functions can be used when the interest is focused on some event properties, but does not concern the trace-context of those events. Example of event-level utility function is the search for LPMs describing process fragments with high financial cost.

Activity defines the utility of an LPM based on the frequency of occurrences of each activity. It can be used when the analyst is more interested in some activities with high impact (e.g. lawsuits, security breaches, etc.). This scope if generally the one used in traditional pattern mining algorithms, allowing for the definition of upper bounds to efficiently prune the search space without loss.

Model this class of utility functions is not log-dependent, but allows the analyst to specify preferences for specific structural properties of the LPM.

Functions on the different scopes can be combined to form composite functions, consisting of component functions on one of the scopes above. The utility of an LPM M over an event log L is denoted $u(L, M)$, and we define as *HU-list* a collection of HU-LPMs sorted in descending order according to their utility, with $|\mathcal{S}|$ the number of HU-LPMs in HU-list $\mathcal{S}$. For a more thorough introduction of HU-LPMs and related concepts, we refer the reader to [22].

4 Pruning Strategies

In contrast to regular Local Process Model (LPM) mining, High Utility LPM (HU-LPM) mining cannot be performed with techniques that prune the search space based on frequency, leading to a large search space. Therefore, there is a need for an alternative pruning strategy for HU-LPM mining that makes mining possible on larger logs, however the utility metric as defined in [22] is not necessarily anti-monotonic, preventing any lossless reduction of the search space. When setting a stopping criterion c on LPMs such that we expand an LPM M only when c holds for M, we say that c is anti-monotonic when M violating c implies that all $M' \in Exp(M)$ violate c. However, this property does not hold for the utility functions defined in [22], as the expansion of an LPM can either have a utility lower or higher than the utility of the LPM where it is an expansion of (Property 1).

Property 1. $\exists_{M \in \mathcal{M}}(\exists_{M' \in Exp(M)} u(L, M') < u(L, M) \wedge \exists_{M' \in Exp(M)} u(L, M') \geq u(L, M))$.

We show that anti-monotonicity does through the following counter-example. Let M_1, M_2, M_3 and M_4 be the four LPMs shown in Figure 1, with $M_2 \in Exp(M_1)$, $M_3 \in Exp(M_2)$, and $M_4 \in Exp(M_3)$. Let event log L consist of the single trace shown in Figure 2a, and let the utility be the sum of the cost attributes of the events that belong to replayable fragments. This results in utilities $u(L, M_1)$=1350, $u(L, M_2)$=2800, $u(L, M_3)$=1300 and $u(L, M_4)$=1400. It is easy to see that utility is not anti-monotonic, as $u(L, M_3) < u(L, M_2)$, but $u(L, M_4) > u(L, M_3)$. This leads to non-optimal HU-LPMs when we prune using a minimum utility threshold, e.g., stopping criterion $c : u(L, M) \geq 1350$ leads to M_3 not being expanded because its utility is below the threshold, while the utility of M_4 would have again been above the threshold. This is mainly explained by the fact that the utility added by the new activity does not compensate for the utility lost because of the fragments that do not fit the new LPM but that did fit the previous LPM.

Definition 2. ***High-Utility Local Process Model Mining Problem:*** *Given an event log L, the HU-LPM mining problem is defined as the task of discovering a set of LPMs with utility above a predefined threshold u_{min}, i.e., $u(L, LPM) \geq u_{min}$.*

In High-Utility Local Process Model (HU-LPM) discovery, the size of the search space grows combinatorially with the number of activities. Reducing the search space is an inevitable step to ensure efficiency or even to enable to algorithm to run in acceptable time. We have shown that the utility metric is not anti-monotonic and that we therefore cannot reduce the search space without loss. However, heuristics can be used to reduce execution time, without formal guarantee of finding an optimal solution; i.e., the discovered set of LPMs fulfilling the utility threshold might be incomplete.

The remainder of this section is as follows. We define new concepts related to HU-LPMs in Section 4.1, we introduce two memoryless heuristics in Section 4.2, and introduce two memory-based heuristics in Section 4.3.

4.1 Basic Concepts

Let $M \in \mathcal{M}$ be a HU-LPM, then $Par(M)$ denotes the parent of m; $Par(M)=M' \in \mathcal{M}$ such that $M \in Exp(M')$. For example, for the process trees of Figure 1, $Par(M_3) = M_2$. We generalize the concept of parent in Equation 1, and define $Par^i(M)$ as the i^{th} parent of M, with $Par^0(M) = M$, $Par^1(M) = Par(M)$, $Par^2(M) = Par(Par(M))$ and so forth; In general, we define $Par^i(M)$ as:

$$Par^i(M) = \begin{cases} Par^{i-1}(Par(M)) & \text{if } i > 1, \\ M & \text{if } i = 0. \end{cases} \quad (1)$$

For example, for the process trees of Figure 1, $Par^3(M_4)=M_1$. Note that $M \notin dom(Par)$ for LPMs $M \in \mathcal{M}$ that are initial LPMs, as initial LPMs have no parent LPM defined. Furthermore, we define $it\_nb(M)$ as the number of expansions to reach HU-LPM M from an initial HU-LPM; e.g. $it\_nb(M_3)=2$. Formally:

$$it\_nb(M) = \begin{cases} 0, & \text{if } M \notin dom(Par), \\ it\_nb(Par(M)) + 1, & \text{if } M \in dom(Par). \end{cases} \quad (2)$$

Using Par and it_nb we define $Anc(M)$ as the set of ancestors of the LPM M; $Anc(M) = \bigcup_{i=1}^{it\_nb(M)} Par^i(M)$. For example for the process trees of Figure 1, $Anc(M_4)=\{M_1, M_2, M_3\}$.

Based on these definitions, we now introduce four heuristics to reduce the execution time of HU-LPM mining.

4.2 Memoryless heuristics

This first type of heuristics focuses on local comparisons, i.e., an LPM is only compared with its parent, the previous expansions are not considered. The heuristics work as follows: for a defined number of successive extensions (noted k, such that $0<k<exp\_max$), the new LPM is allowed to have a utility lower than or equal to the utility of its parent.

For each heuristic, we define a *continuation criterium function*, $ctn(L, k, M)$, which results to 1 if the k most recent expansion steps leading to LPM M meet the requirements of the heuristic, indicating that M should be expanded further. Otherwise, function ctn results to 0, indicating that M should not be expanded further, therefore reducing the search space and speeding up the discovery of HU-LPMs. We introduce heuristic h_1, that formalizes the function $ctn(L, k, M)$ as defined above:

- **Heuristic 1 (h_1)**: The expansion of LPM $M \in \mathcal{M}$ is stopped if all LPMs from the $k-1^{th}$ parent of M to M itself have a utility lower or equal to the utility of its parent.

$$ctn(L,k,M)=\begin{cases} 1-\prod_{i=0}^{min(it\_nb(M),k)-1}(u(L,Par^i(M))\leq u(L,Par^{i+1}(M))), & \text{if } it\_nb(M)\geq 1 \\ 1, & \text{otherwise.} \end{cases} \quad (3)$$

Note that $(u(L,Par^i(M))\leq u(L,Par^{i+1}(M))$ is a boolean expression that evaluates to 1 when true and evaluates to 0 when false. As we want initials LPMs to always be expanded independently of any heuristic, $ctn(L,k,M)=1$ when $it\_nb(M)=0$, and the function defined by each heuristic otherwise. We additionally propose heuristic h_2, a relaxed version of h_1, where the expanded LPM is always allowed to have the same utility as its parent:

– **Heuristic 2 (h_2)**: The expansion of LPM $m \in \mathcal{M}$ is stopped if all LPMs from the $k{-}1^{th}$ parent of M to M itself have a utility strictly lower than the utility of their parents.

$$ctn(L,k,M)=\begin{cases} 1-\prod_{i=0}^{min(it\_nb(M),k)-1}(u(L,Par^i(M))< u(L,Par^{i+1}(M))), & \text{if } it\_nb(M)\geq 1 \\ 1, & \text{otherwise.} \end{cases} \quad (4)$$

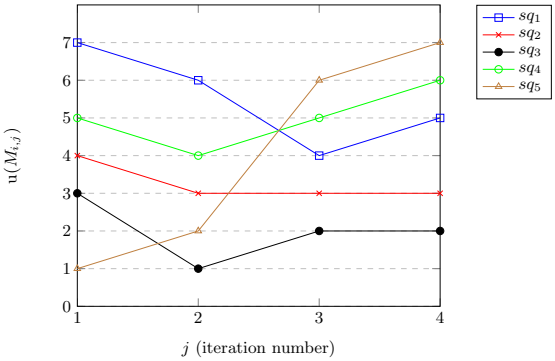
4.3 Memory-based heuristics

The second type of heuristics keeps in memory the set of LPMs produced by the successive expansions. Instead of comparing two successive expansions, it compares an extension with its best ancestor. For an LPM M, we define $B(L,M)$ as the highest utility among the ancestors of M in event log L; $B(L,M){=}u(L,M')$ of LPM $M'{\in}Anc(M)$ such that $\nexists_{M''\in Anc(M)} : u(L,M'){>}u(L,M')$. The heuristics work as follows: for a defined number of successive expansions (noted k, such that $0{<}k{<}exp\_max$), the expanded LPM is allowed to have a utility lower than or equal to the utility of its best ancestor.

We introduce heuristic h_3, that formalizes the function $ctn(L,k,M)$ as defined above:

– **Heuristic 3 (h_3)**: The expansion of LPM $M \in \mathcal{M}$ is stopped if all LPMs from the $k{-}1^{th}$ parent of M to M itself have a utility lower or equal to the highest utility among their ancestors.

$$ctn(L,k,M)=\begin{cases} 1-\prod_{i=0}^{min(it\_nb(M),k)-1}(u(L,Par^i(M))\leq B(L,Par^i(M))), & \text{if } it\_nb(M)\geq 1 \\ 1, & \text{otherwise.} \end{cases}$$



(a)

exp. seq. \ heuristic	h_1	h_2	h_3	h_4
sq_1:	✗	✗	✗	✗
sq_2:	✗	✓	✗	✗
sq_3:	✓	✓	✗	✗
sq_4:	✓	✓	✗	✓
sq_5:	✓	✓	✓	✓

(b)

Fig. 3. *(a)* Four successive expansions on five initial LPMs, *(b)* the pruning scope of the different heuristics.

(5)

We also propose heuristic h_4, a relaxed version of h_3, where the expanded LPM is always allowed to have the same utility as its best ancestor:

– **Heuristic 4 (h_4)**: The expansion of LPM $M \in \mathcal{M}$ is stopped if all LPMs from the $k{-}1^{th}$ parent of M to M itself have a utility strictly lower than the highest utility among their ancestors.

$$ctn(L,k,M) = \begin{cases} 1 - \prod_{i=0}^{min(it\_nb(M),k)-1} (u(L,Par^i(M)) < B(L,Par^i(M))), & \text{if } it\_nb(M) \geq 1 \\ 1, & \text{otherwise.} \end{cases} \quad (6)$$

To illustrate these four heuristics, let the plot in Figure 3a be our running example. Let sq_i represent a sequence of expansions from an initial LPM, and $sq_{i,j}$ be the j^{th} LPM of that sequence of expansions sq_i. For each heuristic and $k = 2$, Figure 3b presents the sequences that would be expanded until the fourth step (marked with ✓) and those that would be stopped before (marked with ✗).

Here, sq_1 and sq_5 are two extremes. While sq_1 contains two successive utility decreases ($sq_{1,2}$ and $sq_{1,3}$), sq_5 contains only LPMs having a higher utility at each expansion. In consequence, every heuristic would have stopped sq_1 after $sq_{1,3}$; removing further expansions from the search space, and would have expanded sq_5 until $sq_{5,4}$. Sq_2 is only expanded until the fourth step by h_2 because this relaxed version allows for the utility to stagnate during 2 steps after a first decrease. On the contrary, the expansion of sq_4 is only stopped by h_3 because the utility

obtained in the first step remains higher than the ones obtained at each further step. Finally, sq_3 is expanded until the fourth step by the memoryless heuristics but stopped by the memory-based heuristics because the increase at $sq_{3,3}$ is enough for the utility of the expansion to be higher than the utility of its parent, but not enough to be at least equal to the utility of its best ancestor.

Heuristic h_2 is the most permissive as an LPM is only compared with its parent and can have the same utility. On the contrary, Heuristic h_3 is the most restrictive as a LPM is compared with its best ancestor and must have a higher utility. As heuristics are approximate methods, some LPMs will be wrongly pruned. In the example in Figure 3, the expansion line sq_4 would have been pruned by heuristic h_1 after $sq_{4,3}$. However, we notice that the LPM produced in the next expansion has a utility higher than the best ancestor. This is an example of expansion line that shouldn't have been stopped. Therefore, a good strategy will be a compromise between the number of good HU-LPMs we allow to loose and how small the search space has become thanks to the pruning methods.

5 Experiments

In this section we evaluate the four heuristic HU-LPM mining approaches. We detail the experimental setup in Section 5.1 and discuss the results in Section 5.2.

5.1 Methodology

We evaluate the four HU-LPM discovery heuristics using three event logs: the BPI'13 closed problems log, consisting of 1487 traces and 6660 events, the BPI'13 open problems log, consisting of 819 traces and 2351 events, and an artificial log used in the Process Mining book [1] (Chapter 1), consisting of 6 traces and 42 events.

For each event log we first apply the HU-LPM discovery algorithm without any pruning, generating the desired list of HU-LPMs in terms of quality and leading to the worst case number of LPMs that are explored. As a next step, we discover HU-LPMs with each of the four different heuristic strategies, and $\{1, 2, 3\}$ the values of parameter k, i.e. the number of expansions allowed not to meet the heuristic utility requirements. We limit the LPM expansion procedure to four successive expansions, i.e., $exp\_max = 4$, to be able to perform the experiments within reasonable time. We compare the number of explored LPMs using the heuristics with the number of LPMs explored when no pruning was applied. We do the same with execution times. A lower fractions of explored LPMs with pruning compared to the number of LPMs explored without pruning represents higher speedup in HU-LPM mining.

As shown in Section 4, the heuristics might prevent the discovery of HU-LPMs with high utility, leading to HU-lists of lower quality. Let $\mathcal{S}_a = \langle M_1, M_2, \ldots, M_n \rangle$ be the HU-list obtained using heuristic a. We define $\mathcal{S}_{id}$ as the ideal HU-list; i.e., the HU-list extracted without any pruning. To assess the efficiency of the four heuristics, we compare the HU-list extracted with the heuristics with the ideal

HU-list obtained with the existing HU-LPM mining technique [22] which does not use any pruning. The quality of the extracted HU-list depends on the utility of the HU-LPMs in the HU-list compared to the utility of the HU-LPMs in the ideal HU-list. We compare the HU-list and the ideal HU-list using the normalized Discounted Cumulative Gain (nDCG) [7], which is an evaluation metric for rankings that is one of the most commonly used metrics in the Information Retrieval field [21]. Discounted Cumulative Gain (DCG) measures the quality of a ranking based on the relevance of the elements in the ranking in such a way that it gives higher importance to the top positions in the ranking. We denote the relevance of the element of the ranking at position i with rel_i. For the evaluation of a HU-list we regard rel_i to be the utility of the LPM at position i. Equation 7 formally defines DCG over the first p elements of a ranking.

$$DCG_p = \sum_{i=1}^{p} \frac{2^{rel_i} - 1}{log_2(i+1)} \tag{7}$$

IDCG is defined as the DCG obtain the optimal ranking, which is the ideal HU-list in our example. Equation 8 defines nDCG based on the DCG of a ranking and the IDCG of the respective ideal ranking.

$$nDCG_p = \frac{DCG_p}{IDCG_p} \tag{8}$$

We limit the nDCG calculation to the p first HU-LPMs in the ranking. The upper bound of p is the cardinality of the HU-list $\mathcal{S}_a$ obtained with pruning, i.e., $0<p\leq|\mathcal{S}_a|$.

5.2 Results

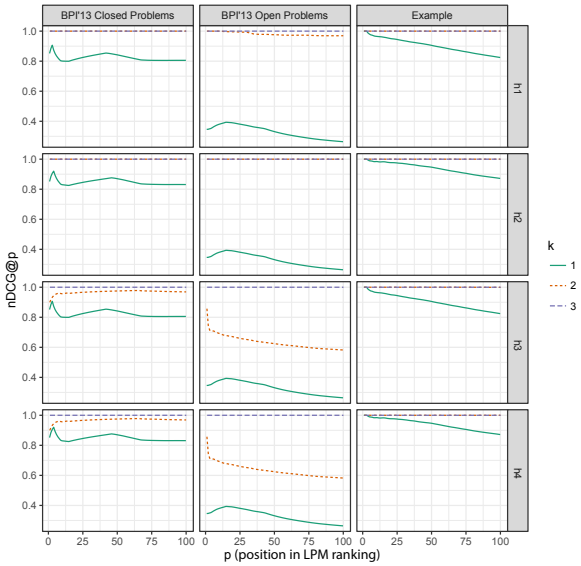
The results of the experiments are shown in Figure 4. Figure 4c shows the nDCG values obtained for the three event logs, the four heuristics and the three values of k. The x-axis of each plot represents the p value used to compute nDCG@p, represented on the y-axis. Table 4a presents the number of LPMs generated by the different heuristics and the mining algorithm without any pruning. We also present in Table 4b the execution times in seconds of each mining. Setting parameter k to 1 leads to the largest reduction of search space for each heuristic, therefore resulting in the highest speedup. Furthermore, the nDCG values show that k=1 still results in the extracton high-quality HU-LPM rankings on two of the three datasets: the BPI'13 closed problems and the example log. The BPI'13 Open Problems dataset however shows that in some cases $k = 1$ leads to overly aggressive pruning, preventing LPMs with high utility from being found, and leading to low quality HU-LPM rankings. For the BPI'13 closed problems log heuristic h_1 with $k = 1$ decreases the search space from 1771 LPMs to 512 LPMs (more than three times less), resulting in a computation speedup of 17x at best (from 11.9s to 0.7s). At the same time heuristic h_1 with $k = 1$ on this log results

Log	no pruning	heur.	k=1	k=2	k=3
Closed	1771	h_1	512	1530	1771
Closed	1771	h_2	527	1580	1771
Closed	1771	h_3	512	1028	1771
Closed	1771	h_4	527	1028	1771
Open	402	h_1	82	356	402
Open	402	h_2	82	372	402
Open	402	h_3	82	224	402
Open	402	h_4	82	224	402
Example	342	h_1	159	324	342
Example	342	h_2	186	331	342
Example	342	h_3	159	299	342
Example	342	h_4	186	316	342

(a)

Log	no pruning	heur.	k=1	k=2	k=3
Closed	28.3	h_1	3.3	17.5	26.9
Closed	28.3	h_2	5.2	20.5	30.3
Closed	28.3	h_3	3.1	11.8	29.1
Closed	28.3	h_4	5.6	19.7	27.4
Open	11.9	h_1	0.7	6.1	10.5
Open	11.9	h_2	0.9	7.3	11.0
Open	11.9	h_3	1.2	3.0	13.5
Open	11.9	h_4	1.2	3.4	12.2
Example	2.9	h_1	0.7	2.7	3.2
Example	2.9	h_2	1.5	2.8	2.9
Example	2.9	h_3	0.7	2.4	3.2
Example	2.9	h_4	1.8	2.8	3.5

(b)



(c)

Fig. 4. *(a)* The number of HU-LPMs generated and *(b)* the execution times (in seconds) per combination of heuristic, event log, and value for parameter k. *(c)* The nDCG results of the four heuristics applied on the three logs.

in an nDCG score of above 0.8 for all values of p, showing that high-quality HU-LPMs are still being found. We observe only minor differences between the four heuristics on the BPI'13 closed problems log and the example log in terms of quality of the HU-list, even if the memory-based heuristics perform better in terms of number of LPMs generated and execution time. However, on the BPI'13 open problems log there is a sizable difference in the quality of the obtained HU-list between heuristics h_1 and h_2 on the one hand and heuristics h_3 and h_4 on the other hand. On this log, heuristics h_1 and h_2 prune only around 10% of the LPM search space with a speed up of 2x in the execution time, and find a HU-list that is very close to the ideal HU-list for k=2. Heuristics h_3 and h_4, however, prune almost half of the search space for k=2 with a speed up of 4x in the execution time, and the resulting HU-list contains a couple of high-utility HU-LPMs, but the nDCG score drops below 0.7 for p=10, indicating that these heuristics where not able to discover more than 10 useful HU-LPMs.

We observe that the speed-up in terms of time is consistently higher than what would be expected based on the share of LPMs that are removed from the search space. We expect that this effect is caused by larger LPMs, generated in later expansion iterations, that are pruned more often than the small LPMs, causing a drop in the average evaluation time per LPM next to the reduction in search space in terms of number of LPMs. Moreover, the computation time of the memory-based heuristics becomes most of the time higher than the computation time of HU-LPM mining without any pruning with k=3, which is caused by the additional comparison procedure that the traditional mining technique does not perform.

6 Conclusion and Future Work

In this paper we have shown that the High-Utility Local Process Model (HU-LPM) mining problem is not anti-monotonic when event-level or trace-level utility functions are used. We introduced four heuristics to reduce the search space and speed up the mining of HU-LPMs. We have shown that the heuristics that we propose still result in the extraction of high-quality rankings of HU-LPMs, while speeding up the mining process up to a factor 17. On larger logs, where mining becomes computationally infeasible without pruning the search space, the proposed heuristics enable the discovery of HU-LPMs.

We have shown in the experiments that the efficiency of the heuristics is log-dependent. In future work, we intend to investigate the properties of the event logs that are responsible for these differences. Based on this we aim at building an automated technique for choosing the appropriate heuristic and the value of k based on the log that results in a good trade-off between computation time of the mining and the quality of discovered HU-LPMs.

References

1. van der Aalst, W.M.P.: Process mining: data science in action. Springer-Verlag Berlin Heidelberg (2016)
2. van der Aalst, W.M.P., Adriansyah, A., van Dongen, B.F.: Replaying history on process models for conformance checking and performance analysis. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 2(2), 182–192 (2012)
3. van der Aalst, W.M.P., Weijters, A.J.M.M., Maruster, L.: Workflow mining: Discovering process models from event logs. IEEE Transactions on Knowledge and Data Engineering 16(9), 1128–1142 (2004)
4. Bergenthum, R., Desel, J., Lorenz, R., Mauser, S.: Process mining based on regions of languages. In: International Conference on Business Process Management. pp. 375–383. Springer (2007)
5. Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: A genetic algorithm for discovering process trees. In: IEEE Congress on Evolutionary Computation. pp. 1–8. IEEE (2012)

6. Buijs, J.C.A.M., Reijers, H.A.: Comparing business process variants using models and event logs. In: Enterprise, Business-Process and Information Systems Modeling, pp. 154–168. Springer (2014)
7. Burges, C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N., Hullender, G.: Learning to rank using gradient descent. In: Proceedings of the 22nd International Conference on Machine Learning. pp. 89–96. ACM (2005)
8. Dave, U., Patel, S.V., Shah, J., Patel, S.V.: Efficient mining of high utility sequential pattern from incremental sequential dataset. International Journal of Computer Applications (2015)
9. van Dongen, B.F., de Medeiros, A.K.A., Verbeek, H.M.W., Weijters, A.J.M.M., van der Aalst, W.M.P.: The ProM framework: A new era in process mining tool support. In: International Conference on Application and Theory of Petri Nets. pp. 444–454. Springer Berlin Heidelberg (2005)
10. Han, J., Cheng, H., Xin, D., Yan, X.: Frequent pattern mining: current status and future directions. Data Mining and Knowledge Discovery 15(1), 55–86 (2007)
11. International Organization for Standardization: ISO/IEC 19505-1:2012 - Information technology - Object Management Group Unified Modeling Language (OMG UML) - Part 1: Infrastructure (2012)
12. Keller, G., Scheer, A.W., Nüttgens, M.: Semantische Prozeßmodellierung auf der Grundlage" Ereignisgesteuerter Prozeßketten". Inst. für Wirtschaftsinformatik (1992)
13. Leemans, M., van der Aalst, W.M.P.: Discovery of frequent episodes in event logs. In: International Symposium on Data-Driven Process Discovery and Analysis. pp. 1–31. Springer (2014)
14. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs containing infrequent behaviour. In: International Conference on Business Process Management. pp. 66–78. Springer (2013)
15. Liesaputra, V., Yongchareon, S., Chaisiri, S.: Efficient process model discovery using maximal pattern mining. In: International Conference on Business Process Management. pp. 441–456. Springer (2015)
16. Maggi, F.M., Mooij, A.J., van der Aalst, W.M.P.: User-guided discovery of declarative process models. In: Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining. pp. 192–199. IEEE (2011)
17. Măruşter, L., van Beest, N.R.T.P.: Redesigning business processes: a methodology based on simulation and process mining techniques. Knowledge and Information Systems 21(3), 267 (2009)
18. Object Management Group: Notation (BPMN) version 2.0. OMG Specification (2011)
19. Srikant, R., Agrawal, R.: Mining sequential patterns: Generalizations and performance improvements. Advances in Database Technology pp. 1–17 (1996)
20. Tax, N., Sidorova, N., van der Aalst, W.M.P., Haakma, R.: Heuristic approaches for generating local process models through log projections. In: 2016 IEEE Symposium on Computational Intelligence and Data Mining. pp. 1–8. IEEE (2016)
21. Tax, N., Bockting, S., Hiemstra, D.: A cross-benchmark comparison of 87 learning to rank methods. Information processing & management 51(6), 757–772 (2015)
22. Tax, N., Dalmas, B., Sidorova, N., van der Aalst, W.M.P., Norre, S.: Interest-driven discovery of local process models. arXiv preprint arXiv:1703.07116 (2017)
23. Tax, N., Sidorova, N., Haakma, R., van der Aalst, W.M.P.: Mining local process models. Journal of Innovation in Digital Ecosystems 3(2), 183–196 (2016)
24. Yin, J., Zheng, Z., Cao, L.: USpan: an efficient algorithm for mining high utility sequential patterns. In: Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 660–668. ACM (2012)

25. Zida, S., Fournier-Viger, P., Wu, C.W., Lin, J.C.W., Tseng, V.S.: Efficient mining of high-utility sequential rules. In: International Workshop on Machine Learning and Data Mining in Pattern Recognition. pp. 157–171. Springer (2015)
26. Zihayat, M., Wu, C.W., An, A., Tseng, V.S.: Mining high utility sequential patterns from evolving data streams. In: Proceedings of the ASE BigData & SocialInformatics 2015. p. 52. ACM (2015)