Goal-oriented Process Mining: A Scalability Experiment

Mahdi Ghasemi
School of EECS
University of Ottawa
Ottawa, Canada
ghasemi75@gmail.com

Daniel Amyot
School of EECS
University of Ottawa
Ottawa, Canada
damyot@uottawa.ca

William Van Woensel
Telfer School of Management
University of Ottawa
Ottawa, Canada
wvanwoen@uottawa.ca

***Abstract*—Process mining exploits event logs to discover an organization's as-is process models, which is an important requirements engineering activity. However, process mining often results in complex "spaghetti-like" models that are difficult to interpret. Different mechanisms can support making sense of such complex models, such as filtering process instances (traces) before feeding them to process mining tools. This paper experiments with one particular approach called *Goal-oriented Processes Mining* (GoPM). Given process goals of interest, GoPM keeps only log traces that satisfy these goals to an expected level. This allows mining a more focused process from the filtered log, i.e., which only captures behavior that satisfies the goals in question. Three algorithms are implemented, and their scalability is assessed through six experiments involving a total of 1960 runs over synthetic event logs. The experiments' runtimes show that the algorithms are feasible and scalable enough to apply to large event logs, which indicates their practicality.**

***Index Terms*—Process Mining, Goal Modeling, GRL, GoPED, Binary Optimization**

I. Introduction

Process mining is an approach to discover as-is processes in organizations, based on data-driven evidence in the form of event logs. Such logs capture the "footprints" of processes, also known as events, in the organization's Information Systems (IS). This provides a more realistic view of the process as opposed to relying on the (potentially biased or incomplete) perception of process participants or owners [1]. The discovered process models, often represented as Directly-Follows Graph (DFGs), Petri Nets, or Business Process Model and Notation (BPMN) models [2], with quantitative annotations (frequencies, delays, etc.), enable requirements engineers to analyze processes for performance, cost, compliance-related or other issues, and assess potential improvement, redesign, and automation opportunities [3], [4].

Goal modeling, on the other hand, is a requirements engineering (RE) approach mainly used to capture actors, their goals, and their relationships, in order to analyze what-if and trade-off situations in support of decision making [5]. Whereas process models focus on *who*, *what*, *where*, and *how* aspects of as-is organization processes, goal models mainly focus on *why* and *how much* aspects. Goal models and process models thus provide complementary perspectives that can benefit many areas, including process mining [6].

One major issue commonly faced in process mining is the complexity of the resulting process models, often figuratively characterized as "spaghetti-like", which hinder their interpretability and thus their usefulness. Such complexity derives exactly from its data-driven nature—event logs, often extracted from multiple organization IS, may contain many different types of process instances ("traces"), at times resulting from a lack of process structure, process variability, noise, truncated traces, and other such reasons [1]. Different mechanisms exist to mitigate such complexity issues [7], [8], including:

- Exclusion of truncated traces, which started before or finished after the event log time window;
- Noise filtering, to eliminate infrequent activities or traces;
- Trace filtering, along attributes such as region, dates, or product;
- Clustering, to decompose a model into separate models based on similarity metrics at the trace or activity levels;
- Abstraction, where some simple activities are aggregated into more abstract ones;
- Object-centric modeling, where an object type (e.g., an order or invoice) follows its own path and cross-object relations are shown explicitly.

This paper explores a filtering technique that leverages the complementary nature of goal modeling and process mining, called *Goal-oriented Process Mining* (GoPM, see Fig. 1). Here, process traces are selected that meet goals of interest, at given satisfaction levels, within a goal model. Subsequently, a focused and simplified process can be mined from the filtered log, capturing behavior that satisfies the goals in question. In other words, GoPM is able to improve the *rationality* of discovered models, i.e., their alignment with desired goals.

In order to remain independent from particular process mining tools, GoPM performs such filtering on the input event log itself. In detail, given an event log that contains indicator attributes, a goal model, and desired goal criteria (i.e., goals with desired satisfaction levels), GoPM selects process *variants*[1] in the log that satisfy the goal criteria, and produces a

This work was supported by NSERC Discovery and PGS-D grants.

[1]A process variant is a unique sequence of activities in a process, and can cover multiple process instances (traces).

filtered log. GoPM handles non-trivial criteria for aggregating sets of traces dynamically (e.g., based on averaged values for specific goals or the entire goal model), something that can be solved through binary optimization [9].

This paper builds on previous work on GoPM that introduced a first version of three selection algorithms for process variants, collectively named *Goal-oriented Process Enhancement and Discovery* (GoPED) [9], with preliminary tool support for preparing the traces needed as input to these selection algorithms [10]. This paper contributes a revised version of these algorithms, their Python-based implementation, and a performance assessment focused on execution times as input event logs become larger and more complex. Our research question here is: *How fast can the* GoPED *algorithms select traces from event logs when the complexity of the log varies along diverse factors?* Such factors include the number/length of cases/traces, their distribution, the number of goals, and goal satisfaction levels (criteria boundaries).

In this paper, Sect. II first gives an overview of GoPM followed by an illustrative example in Sect. III. Section IV describes the three GoPED algorithms, which are then assessed in Sect. V for scalability against six event log factors. Section VI describes related work, whereas Sect. VII presents limitations and threats to validity. Finally, Sect. VIII provides our conclusions.

II. Goal-oriented Process Mining

The main input of conventional process mining activities is the event log, resulting from the execution of processes and extracted from different IS. A log includes a number of events, which are minimally described by an instance/trace/case identifier, the event/activity name, and timestamps, but can also have other attributes (e.g., resources and indicators). Process mining techniques have been growing into activity-focused approaches that do not typically consider goals pursued by individual cases and satisfaction levels for different stakeholders' goals [11]. This situation can threaten the *rationality* of discovered models, which adds to the complexity of already hard-to-read "spaghetti-like" models that often result from process mining. Although such complex models reflect reality, they do not distinguish between traces that pursue different goals (e.g., processing time vs. outcome quality), or traces that are misaligned with any relevant goal [12]. This lack of a goal focus, especially in flexible environments that allow many alternatives during process execution, has to currently be manually dealt with by process mining practitioners.

A literature review of the intersection of goal modeling and process mining [11] showed that, although both research areas are growing, few studies are conducted at their intersection. The area of *intention mining* [13], while related to this intersection, is more about mining process intentions than being a goal-based filtering technique per se. The combination of goal modeling and process mining can achieve synergetic effects that augment the rationality and focus of the discovered process models and eventually improve the satisfaction of process stakeholders, especially in an RE context [14].

GoPM was proposed to partially address the above research gap. GoPM is a process mining approach concerned with both the sequencing of activities on the one hand, and processes' goals and satisfaction indicators on the other. GoPM includes data pre-processing steps (blue box on the left of Fig. 1) to enhance the input event log: it first extracts traces from the event log, and then adds satisfaction levels of corresponding goals (from a goal model) to the extracted traces as additional attributes [10]. GoPED algorithms then exploit these new attributes (red box on the right of Fig. 1) to select the traces that meet certain goal criteria [9]. The selected traces are then transformed again to an event log before feeding the latter to a conventional process mining tool, which generate a corresponding process model (DFG, BPMN, Petri Nets, etc.).

Many goal modeling languages have been proposed to support RE activities [5]. GoPM uses one of them, namely the Goal-oriented Requirement Language (GRL), part of the User Requirements Notation international standard [15]. GRL support different types of goals and goal relationships (AND/OR/XOR decomposition, contributions, and dependencies), together with qualitative and quantitative satisfaction propagation mechanisms [16]. But more importantly, GRL includes an *indicator* concept [15], [17], [18] that can convert values from event logs into normalized satisfaction levels. These levels are propagated to the rest of the goal model, and then goal satisfaction values are used by GoPM to select goal-satisfying traces.

III. Illustrative Example

A simplified *diagnosis of gestational diabetes* (DGD) process from healthcare is used here to illustrate GoPM and its algorithm. An event log of 10 patients (cases) who went through the DGD process is shown in Table I. We use short names to encode the activities: a = admission, b = regular test, c = check the result, d = request for advanced test, e = advanced test, f = request for repetition, and g = send the result. There are five different variants in this event log, each covering different numbers of cases, which are separated by horizontal lines. Extra case attributes: process duration in days, its overall cost, patient satisfaction rating (on a 1-to-10 scale, where 10 is the best rating), and accuracy of the result (1 = correctly diagnosed, and 0 = incorrectly diagnosed).

Figure 2(a) shows the Directly-Follows Graph (DFG) representation of this entire event log, mined using the Apromore tool [19]. This visualization also shows the total frequencies of activities and their transitions, as well as average times between activities. With more realistic (and longer) logs, such models often become too complex to understand, hence the need for mechanisms such as abstraction and filtering.

Let us now assume a simple GRL model (Fig. 3) for the DGD process that shows the main patient satisfaction goal (G6) and how it is decomposed. Indicators (doubled-lined hexagons), assessed through the extra attributes of the event log (Table I), contribute the leaf goals. For example, patient rating contributes fully (100%) to the satisfaction of G3, which in turn is a partial contributor (at 25%) to G5.

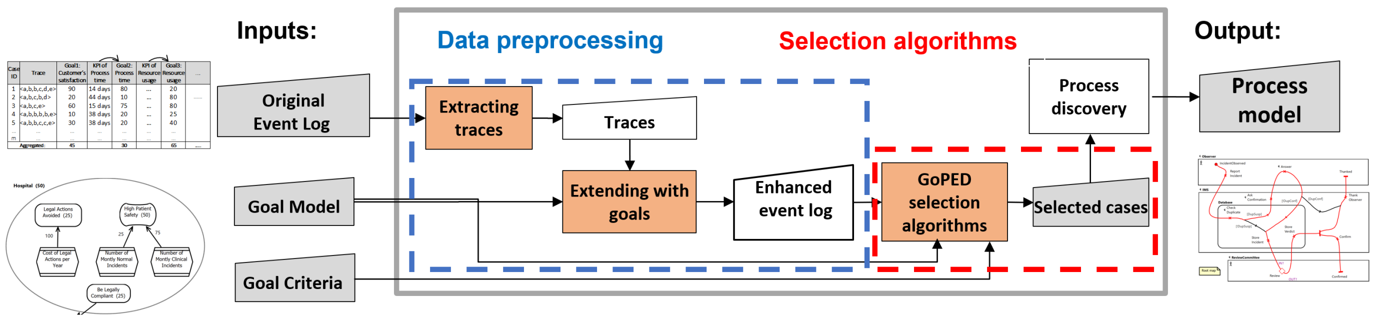
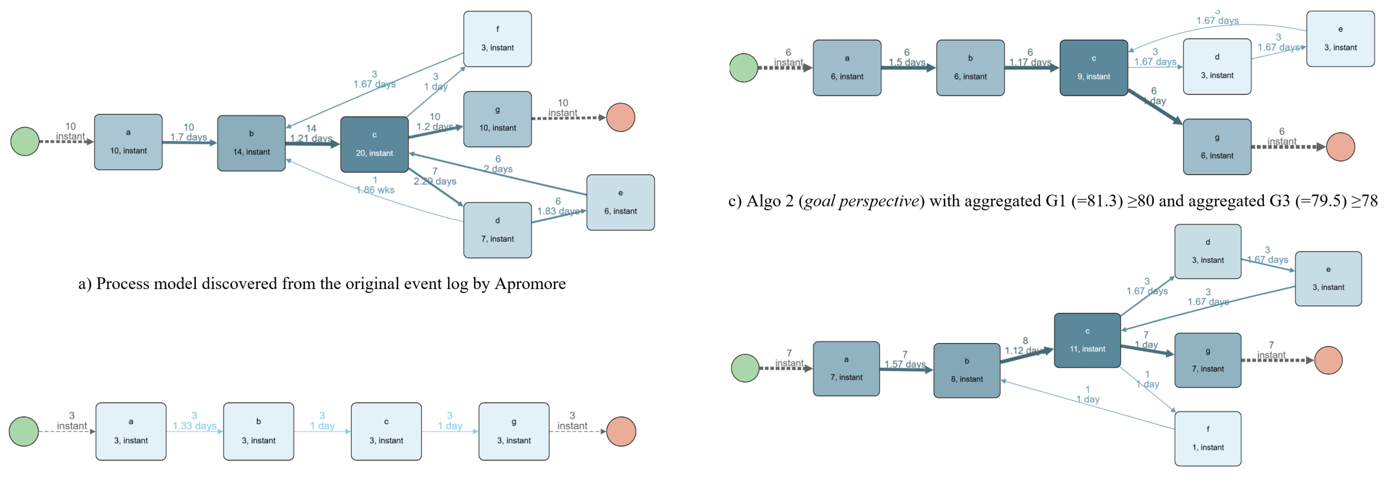


Fig. 1. Overview of GoPM, including its GoPED algorithms.



a) Process model discovered from the original event log by Apromore

b) Algo 1 (*case perspective*) with G1≥80, G4≥100, and confidence = 60%

c) Algo 2 (*goal perspective*) with aggregated G1 (=81.3) ≥80 and aggregated G3 (=79.5) ≥78

d) Algo 3 (*organization perspective*) with comprehensive satisfaction (=66.1) ≥ 65

Fig. 2. Initial process model (a), with models illustrating the three algorithms (b-d), all shown as Directly-Follows Graphs using Apromore.

TABLE I
DGD EVENT LOG FOR 10 PATIENTS, WITH CURRENT DATA ATTRIBUTES.

Case	Trace	Days	Cost	Rating	Accuracy
C_1	$\langle a, b, c, g\rangle$	4	400	9	1
C_2	$\langle a, b, c, g\rangle$	5	400	9	1
C_3	$\langle a, b, c, g\rangle$	5	400	9	0
C_4	$\langle a, b, c, d, e, c, g\rangle$	11	850	8	1
C_5	$\langle a, b, c, d, e, c, g\rangle$	9	850	7	1
C_6	$\langle a, b, c, d, e, c, g\rangle$	10	850	8	1
C_7	$\langle a, b, c, f, b, c, g\rangle$	8	600	7	1
C_8	$\langle a, b, c, f, b, c, d, e, c, g\rangle$	17	1100	6	1
C_9	$\langle a, b, c, f, b, c, d, e, c, g\rangle$	16	1100	5	1
C_10	$\langle a, b, c, d, b, c, d, e, c, g\rangle$	31	1150	4	1

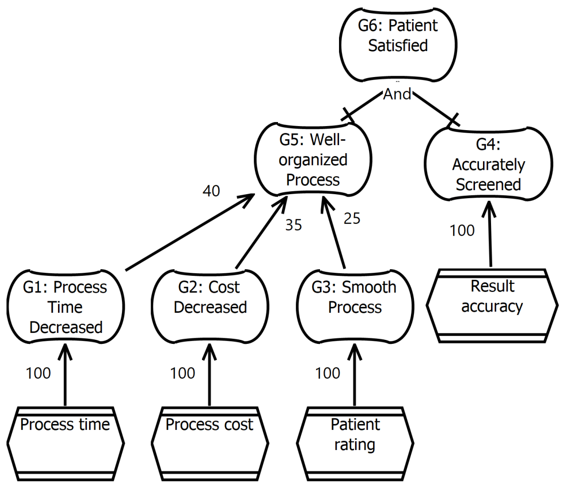


Fig. 3. Goal model of the DGD process.

In GRL, indicators (a.k.a. KPIs) have three parameter values against which an observable value is compared: the target (satisfaction = 100), the threshold (satisfaction = 50), and the worst (satisfaction = 0). Satisfactions levels are interpolated linearly between the target and threshold values, and between the threshold and worst values [15], [17]. For our DGD goal model, these indicator parameters are defined in Table II.

For example, using Fan et al.'s arithmetic formula generation tool for GRL models [18], the process cost indicator function would be as follows:

$$Process\_Cost(C) = \begin{cases} 100 & \text{if } C \leq 400 \\ 0 & \text{if } C \geq 1200 \\ \left|\dfrac{950 - C}{950 - 400}\right| \times 50 + 50 & \text{if } 400 < C \leq 950 \\ -\left|\dfrac{950 - C}{950 - 1200}\right| \times 50 + 50 & \text{if } 950 < C < 1200 \end{cases}$$

TABLE II
KPI DEFINITIONS FOR THE DGD PROCESS.

Indicator	Linked Goal	Worst v.	Threshold v.	Target v.
Process time (days)	Process time decreased	35	13	4
Process cost ($)	Cost decreased	1200	950	400
Patient rating	Smooth process	1	6	10
Result Accuracy	Accurately screened	0	–	1

The initial event log (Table I) can then be augmented with additional columns where the satisfaction level of each goal has been computed from the indicators. For example, for case C_7 (where the cost is 600, see Table I), $Process\_Cost(600) = 82$. After computing the indicators for all the cases, we can populate the satisfaction values for the goal columns (Table III), leading to an enhanced log that can be filtered on these values.

TABLE III
ENHANCED LOG OF THE DGD PROCESS: ADDITIONAL GOAL SATISFACTION LEVELS, WITH AGGREGATED VALUES.

Case	G1	G2	G3	G4	G5	G6
C_1	100	100	88	100	97	97
C_2	94	100	88	100	95	95
C_3	94	100	88	0	95	0
C_4	61	59	75	100	64	64
C_5	72	59	63	100	65	65
C_6	67	59	75	100	66	66
C_7	78	82	63	100	76	76
C_8	41	20	50	100	36	36
C_9	43	20	40	100	34	34
C_10	9	10	30	100	15	15
Aggregate satisfaction:	**65.9**	**60.9**	**66**	**90**	**64.1**	**54.8**

Adding goal models in a process mining context enables GoPM to discover simpler and understandable process models that meet measurable goal-oriented criteria. The models generated through GoPM can help organizations understand and promote good practices (e.g., by reinforcing traces/behavior that meet certain goals), and help people avoid process options that lead to underperforming goal trade-offs (e.g., through training, controls, automation, or improved processes).

IV. GoPED ALGORITHMS AND IMPLEMENTATION

This paper provides slight improvements over a first version of the three Goal-oriented Process Enhancement and Discovery (GoPED) algorithms presented in [9], together with an implementation and an empirical scalability evaluation.

GoPED exploits goal models to filter event logs in such a way as to keep the variants that meet certain goals. **Algorithm 1** focuses on a *case perspective*, whereby the cases in a variant need to satisfy a set of *goal criteria* (goal, comparison operator, and value). If the proportion of a variant's cases that

Algorithm 1: Case_Perspective

```
Input: EnhancedLog ; // Log enhanced with goals
Input: $Q_{case}$: Set(criteria) ; // Each criterion is a
       triple <goal, operator, value>
Input: conf: number ; // Confidence level
Output: CasesKept: Set(cases)
1  SortByTrace(EnhancedLog);
2  $NumCases \leftarrow$ NumberOfCases(EnhancedLog);
3  $trace(case_0) \leftarrow \langle\rangle$ ; // Add empty trace before log
4  $trace(case_{NumCases+1}) \leftarrow \langle\rangle$ ; // ... and after log
5  $CasesKept \leftarrow \emptyset$;
6  $index \leftarrow 1$;
7  while $index \leq NumCases$ do
8      $SameTraceC \leftarrow \emptyset$ ; // Cases with same traces
9      $NumSatCasesOfTrace \leftarrow 0$;
10     repeat
11         $SameTraceC \leftarrow SameTraceC \cup \{case_{index}\}$;
12         if $case_{index}$ meets all criteria of $Q_{case}$ then
13             NumSatCasesOfTrace++;
14         index++;
15     until $trace(case_{index}) \neq trace(case_{index-1})$;
16     if $NumSatCasesOfTrace/|SameTraceC| \geq conf$ then
           ; // Keep case if confidence level is met
17         $CasesKept \leftarrow CasesKept \cup SameTraceC$;
18 return CasesKept;
```

meet these criteria is higher or equal to an input *confidence* level, then the variant will be kept; else, it is discarded. This can help tolerate some amount of noise in the log.

Based on our example enhanced log (Tables I and III), Fig. 2(b) shows the DFG model discovered on the case variants where $G1 \geq 80 \wedge G4 \geq 100$, with a confidence of 60%. For the first variant (cases C_1 to C_3), 2 out of 3 cases (67%) satisfy these goal-oriented criteria, which meets the required confidence level. None of the other variants meet these criteria, and they are hence filtered out of the event log. This models the behavior that sufficiently satisfies two of the goals in the GRL model. Note that with a confidence level higher than 67%, Algorithm 1 would return an empty set of cases (e.g., an empty model) for this example.

Algorithm 2 focuses on a *goal perspective*, which finds the largest subset of variants that collectively satisfy goal criteria, each with their minimum satisfaction threshold. **Algorithm 3** handles the *organization perspective*, which finds the largest variant subset that meets the entire goal model. These are not trivial problems: adding a variant and its cases to the subset might greatly help to satisfy the criterion related to one goal, but, at the same time, harm satisfying the criterion related to other goals. Both algorithms use constraint-based optimization; after, a case i is kept if x_i evaluates to 1. To either keep or exclude all the cases of a variant (all-or-none rule), we check whether the cases' traces involve identical events, i.e., their traces are equal; if so, their x_i values are also made equal. The algorithms differ in the scope of goals: specific goals (Algorithm 2) or the entire model (Algorithm 3), and this is reflected in the optimization functions used.

For example, using Algorithm 2, the query could be all variants where, when aggregated together (e.g., using an *average*

Algorithm 2: Goal_Perspective

```
Input: EnhancedLog ; // Log enhanced with goals
Input: $Q_{\text{goal}}$: Set(criteria) ; // <goal, threshold>
Input: $G$ ; // Aggregation funct., one per goal
Output: CasesKept: Set(cases)
1 $m \leftarrow$ NumberOfCases(EnhancedLog) ; // NumCases
2 CasesKept $\leftarrow \emptyset$;
3 Solve this binary optimization ; // $x_i$: when equal to
    1, keep case $c_i$ ; $s_{i,j}$: satisfaction level of
    goal $j$ for case $c_i$
4   Maximize z = $\sum_{i=1}^{m} x_i$ s.t.
    $\forall r, t,\ 1 \leq r < t \leq m$ : // All-or-none rule
        trace$(c_r)$ = trace$(c_t) \Rightarrow x_r = x_t$
    // Ensure that $Q_{\text{goal}}$ constraints are met
    $\forall j$ where $G_j \in G : \frac{\sum_{i=1}^{m} x_i \cdot s_{i,j}}{\sum_{i=1}^{m} x_i} \geq \text{threshold}_j$
    $x_i \in \{0,1\}$ // Two potential values for $x_i$
5 for $i = 1$ to $m$ do
6     if $x_i = 1$ then
7         CasesKept $\leftarrow$ CasesKept $\cup \{c_i\}$
8 return CasesKept;
```

Algorithm 3: Organization_Perspective

```
Input: EnhancedLog ; // Log enhanced with goals
Input: $Q_{\text{comp}}$: <oper $\in \{\leq, =, \geq\}$, val $\in$ [0..100]>
Input: $G$ ; // Goal model function
Output: CasesKept: Set(cases)
1 $m \leftarrow$ NumberOfCases(EnhancedLog) ; // NumCases
2 CasesKept $\leftarrow \emptyset$;
3 Solve this binary optimization ; // $x_i$: when equal to
    1, keep case $c_i$ ; $s_{i,j}$: satisfaction level of
    goal $j$ for case $c_i$
4   Maximize z = $\sum_{i=1}^{m} x_i$ s.t.
    $\forall r, t,\ 1 \leq r < t \leq m$ : // All-or-none rule
        trace$(c_r)$ = trace$(c_t) \Rightarrow x_r = x_t$
    // Ensure that $Q_{\text{comp}}$ constraint is met
    $G(\frac{\sum_{i=1}^{m} x_i \cdot s_{i,1}}{\sum_{i=1}^{m} x_i}, \ldots, \frac{\sum_{i=1}^{m} x_i \cdot s_{i,n}}{\sum_{i=1}^{m} x_i}) < oper >< val >$
    $x_i \in \{0,1\}$ // Two potential values for $x_i$
5 for $i = 1$ to $m$ do
6     if $x_i = 1$ then
7         CasesKept $\leftarrow$ CasesKept $\cup \{c_i\}$
8 return CasesKept;
```

function, as specified in our algorithm), $G1 \geq 80 \wedge G3 \geq 78$. Note that there is no confidence level in this algorithm. The largest subset of variants that satisfies that query combines the first and second variants (cases C_1 to C_6), with average values $G1 = 81.3$ and $G2 = 79.5$. The process model discovered from these variant is shown in Fig. 2(c). Adding any other variant would dissatisfy one or both goals.

Finally, using Algorithm 3, a sample query would be to include all variants whose aggregate (average) goal model satisfaction is at least 65. The largest combination of variants that satisfies that query includes cases C_1 to C_7, with an average model satisfaction ($G6$) of 66.1. The model discovered on these variants is shown in Fig. 2(d).

Our GoPM implementation (in Python) is composed of four complementary components, all available online[2]. As shown in Fig. 1, the first component (TraceMaker) extracts traces (cases) from an original log. The second component (EnhancedLogMaker) augments these traces with KPI and goal satisfaction values, as defined in an input GRL goal model. The third component implements the three GoPED algorithms, and uses IBM CPLEX and DOcplex.MP[3] for the optimization part. Finally, the fourth component (EventLogRefiner) filters the original event log by only keeping the cases selected by GoPED. The first two components were already presented in [10]; the implementation of the last two is introduced here.

V. Scalability Experiments and Results

A. Overview

Our research question was described in the introduction. In our experiment, we use runtime as a key performance indicator to evaluate the algorithms' sensitivity to four event log factors: (L1) distribution of cases among variants, (L2) number of cases, (L3) number of traces, and (L4) length of traces; and two goal factors, (G1) number of goal criteria, and (G2) the goal criteria's boundaries on satisfaction levels. Times were measured on an Intel Core i7-2760QM CPU at 2.40 GHz, 8 GB RAM, and Windows 7.

Given that it is impossible to find real event logs that would enable such detailed evaluation, we generated six sets of synthetic event logs (using Python scripts), one for evaluating each separate factor. Each set comprises event logs that vary along the relevant factor, while the other five remain constant (except for factors related to the distribution and the number of traces, which are partially dependent). The first four experiments used the following constant goal criteria:

- Algo 1: $G1 \geq 80\% \wedge G2 \geq 75\%$ with 70% confidence
- Algo 2: $G1 \geq 80\% \wedge G2 \geq 75\%$
- Algo 3: $Q_{comp} \geq 70\%$

More details about these experiments and their results can be found online [20].

B. (L1) Distribution of Cases Among Traces

The cases in an event log can be distributed over different traces (variants), and such different distributions may have an impact on the performance of our algorithms.

A useful, heuristic way to generate distributions of cases over the traces could benefit from the capabilities of the *Beta distribution* from probability theory [21]. This distribution is defined over $[0, 1]$. There are two parameters, $\alpha, \beta > 0$, which control the shape of the distribution and determine if the distribution has one mode and whether it is symmetrical.

Figure 4 shows the Probability Density Functions (PDF) used in our experiment. The PDF determines the likelihood of a random variable having a value within a specific range on the X-axis. We thus divide the X-axis interval [0-1] into

[2]https://github.com/Mahdi-Ghasemi/

[3]https://ibmdecisionoptimization.github.io/docplex-doc/mp/index.html

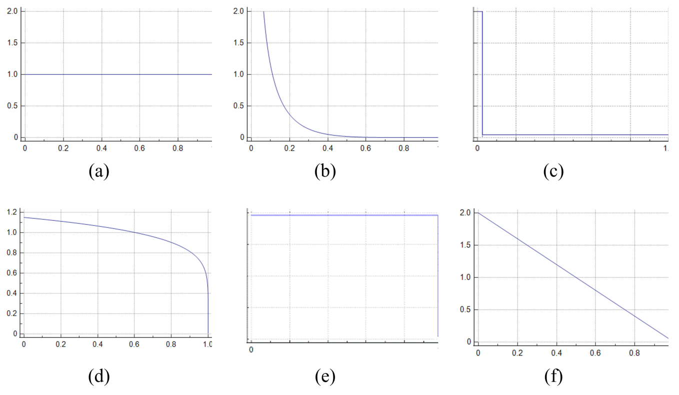


Fig. 4. PDFs used for assessing the influence of distributions.

a number of bins, one for each trace $t_1, t_2, ..., t_{num\_traces}$. Then, we can use the PDF to determine the likelihood of a case (random variable) being assigned to a trace (X-axis bin).

(a) *Beta(1, 1)*, the Uniform distribution where the same number of cases is assigned to each trace.
(b) *Beta(0.2, 6)*, which is highly skewed; a large number of cases is assigned to a small number of traces, and a small portion of cases is assigned to a large portion of traces.
(c) *Maximum Variance* distribution – an extreme case of (b) – where all traces except one have only a single case, and the remaining trace has all the remaining cases.
(d) *Beta(1, 1.15)*, where the distribution makes a concave-down curve.
(e) Another extreme case assigns only one case to one trace, and all remaining traces are equally assigned to the remaining cases.
(f) *Beta(1, 2)*, a triangle-shaped distribution – a middling between (b) and (d).

The constant factors are 50,000 cases and 1000 traces of length 6 (i.e., including 6 events). We created 6 separate event logs, one per distribution (a)-(f), and applied the algorithms on each event log 20 times. The averages are reported here.

Results. In the end, there was no meaningful trend for the runtime when the distribution of cases among traces changes, for all three algorithms. Based on a one-way ANOVA test with a significance level of 0.05, the null hypothesis "the means of all six groups are equal" is rejected for Algorithm 1, i.e., the six groups of runtimes do not have the same mean. Practically, the average time of 20 runs for distribution (e) is 0.276 s (as the maximum), while the average time of 20 runs for (c) is 0.248 s (as the minimum). The difference between those two runtimes is only 11% (i.e., 0.028 s), which does not suggest a significant correlation between the runtime and the distribution of cases among traces. The same situation holds for the two other algorithms as well.

C. (L2) Number of Cases

Here, as the number of cases reflects the size of the data log that the algorithm should process, we anticipate that this factor positively correlates with runtime.

We vary the number of cases along a logarithmic progression: 50×10^x where $x = 1, 4/3, 5/3, \ldots, i/3, \ldots, 4$. This leads to these numbers of cases: 500, 1077, 2321, 5000, 10772, 23208, 50000, 107722, 232079, 500000.

The constant factors include 500 traces of length 6 with a *Beta(0.2, 6)* distribution. We created 10 event logs, one for each number of cases, and the algorithms were used on each event logs 10 times. The averages are reported here.

Results. As Fig. 5 shows, the runtime of all three algorithms increases with the number of cases. Figure 5(a) shows a linear correlation between the number of cases and runtime for Algorithm 1 (P-value is almost zero). The coefficient of determination (R^2) is 0.9994, meaning 99.94% of the variance in the runtime can be explained by the number of cases. Figure 5(b) shows that a quadratic polynomial curve is a good fit here, with $R^2 = 0.9993$ for Algorithm 2. A similar positive correlation ($R^2 = 0.9999$) applies for Algorithm 3 in Fig 5(c).

Comparing the maximum runtime of each algorithm that happens when the number of cases is 500,000 reveals that with the same event log, the runtime of Algorithm 3 is about two times longer than Algorithm 2 and about 300 times longer than Algorithm 1. Algorithm 1 is a straightforward algorithm that uses two nested loops, but the other two algorithms use CPLEX to solve expensive optimization problems.

D. (L3) Number of Traces (Variants)

In this experiment, the main goal is to find how runtimes change when the number of traces/variants changes.

We vary the number of traces along a linear progression $(1 + i \times (50000 - 1)/9; i = 0, 1, \ldots, 9)$, leading to 10 numbers of traces: 1, 5556, 11112, 16667, 22223, 27778, 33334, 38889, 44445, and 50000 traces. The fixed factors include 50000 cases, traces of length 6, with a *Beta(0.2, 6)* distribution. The algorithms were applied on each of the resulting 10 event logs 10 times, and the averages are reported here.

Results. The runtime of Algorithm 1 (y, in seconds, not visualized here) has a strong quadratic correlation with the number of traces (x), namely $y = 5 \times 10^{-9}x^2 - 7 \times 10^{-5}x + 0.5166; R^2 = 0.994$. While the correlation between the two variables is positive for Algorithm 1, for the other algorithms, the correlation is *negative*. A linear trendline with a negative slope ($y = -8 \times 10^{-5}x + 7.0305$) with a high $R^2 = 0.9685$ suggests that the runtime decreases when the number of traces increases. For Algorithm 3 however, $R^2 = 0.3843$ suggests a weaker correlation.

The negative trend slopes for Algorithms 2 and 3 are not surprising because when the number of traces increases, the number of cases with the same trace decreases, and, in turn, the number of constraints made to handle the all-or-none rule decreases. It is noteworthy that for every *n* cases with the same trace, *n-1* constraints will be generated.

E. (L4) Length of Traces (Variants)

Traces, i.e., variants, are the sequences of activities recorded in logs. Our GoPED algorithms deal with the traces as string

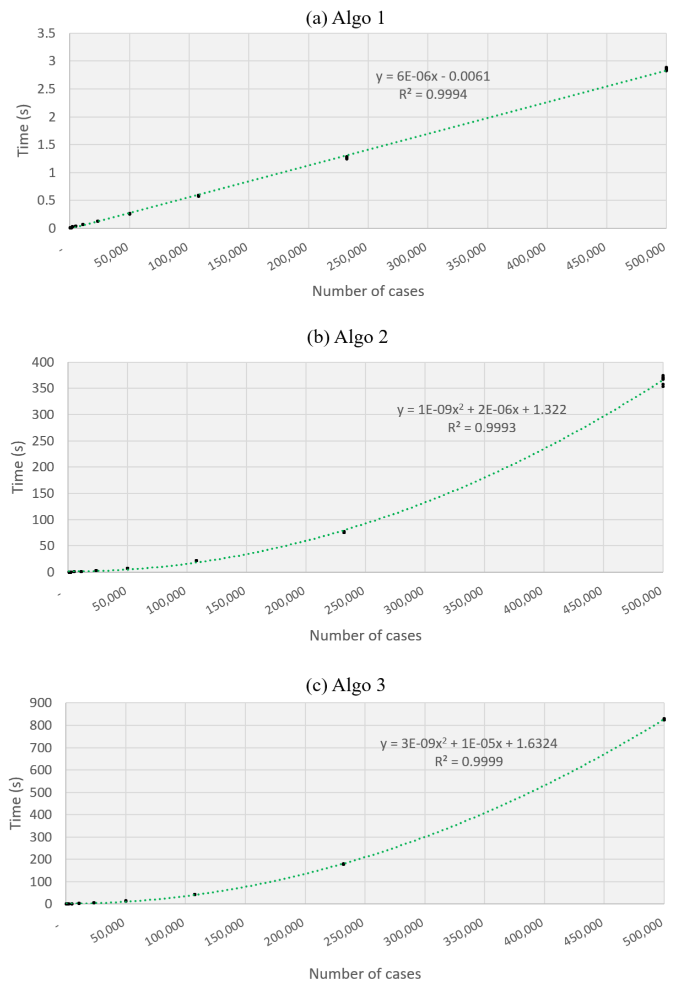


Fig. 5. Runtimes for different numbers of cases.

variables. In all three algorithms, the log should be sorted based on its traces, and there are steps where the traces are compared (e.g., Algorithm 1: lines 1 and 15). Typically, the time needed for comparing and sorting strings depends on the number of characters that should be scanned. In real-world data logs, the traces are usually long. In this step, the main goal is thus to find how the runtime changes when the length of traces changes.

We vary the length of traces along a linear progression $(6+i\times(50000-6)/9; i=0,1,\dots,9)$, leading to 10 lengths: 6, 561, 1116, 1671, 2226, 2780, 3335, 3890, 4445, 5000 activities. The constant factors include 50,000 cases and 5000 traces, with a *Beta(0.2, 6)* distribution. The algorithms were used on the resulting 10 event logs 10 times, and averages are reported here.

Results. There is a strong positive linear correlation between the runtime (y, in seconds) and the length of traces (x) for all three algorithms:

1) Algorithm 1: $y = 0.0011x + 0.3308; R^2 = 0.9996$
2) Algorithm 2: $y = 0.0011x + 6.4472; R^2 = 0.9981$
3) Algorithm 3: $y = 0.0011x + 13.7; R^2 = 0.9977$

Unsurprisingly, the runtime increases when the length of traces increases. It is noteworthy that the length of traces does not impact the dimensions of the optimization problem generated for Algorithms 2 and 3. The only algorithm steps that are impacted by the trace lengths are about sorting the traces and performing comparisons between two traces.

F. (G1) Number of Considered Goals

In Algorithms 1 and 2, a set of goals with satisfaction levels (goal criteria) are input. In Algorithm 1, the satisfaction level of each case will be checked against the criteria. Therefore, for each case, the number of comparisons equals the number of considered goals. In Algorithm 2, each considered goal makes a new constraint for the optimization problem. Therefore, it makes sense to consider the impact of the number of considered goals on the runtime. In Algorithms 3, the goal model is used to calculate the comprehensive satisfaction level from aggregated satisfaction level of all considered goals. Therefore, the number of considered goals for this algorithm represents the size of the goal model.

In this experiment, we generated 10 event logs with 50,000 cases and 1,000 traces of length 6, along a *Beta(0.2, 6)* distribution. The variable part was the number of goals in the goal model, from 1 to 10, with the following criteria:

- Algo 1: $\geq 80\%$ for all considered goals, 70% confidence
- Algo 2: $\geq 80\%$ for all considered goals
- Algo 3: $Q_{comp} \geq 80\%$

Results. For Algorithm 1, we observed a very low linear coefficient between the runtime (y, in seconds) and the number of considered goals (x), namely $y = 0.0069x + 0.4783; R^2 = 0.6026$. This suggests that the number of goals has a positive but very small impact on the runtime. When the number of considered goals increases, Algorithm 1 makes more comparisons for each case, but checking the criteria for each case will return false at the first considered goal whose satisfaction level does not meet the criterion.

Meanwhile, the results for Algorithm 2 ($y = 1.7753x + 4.096; R^2 = 0.9509$) and Algorithm 3 ($y = 1.9362x + 7.4548; R^2 = 0.9917$) suggest a positive strong linear correlation between the two variables. Therefore, increasing the number of considered goals does not significantly impact the runtime of Algorithm 1, but it substantially increases the runtimes of the two other Algorithms.

G. (G2) Goal Criteria's Boundaries

In this last experiment, the goal criteria's boundaries for satisfaction levels (in addition to the confidence level for Algorithm 1) are explored to assess whether the boundaries that mathematically impact the number of selected cases will also impact the runtime.

The constant factors for the event logs here include 50,000 cases, 1000 traces of length 6, a *Beta(0.2, 6)* distribution, and two goal comparisons. For all algorithms, we vary the criteria's boundaries for both goals along a linear progression $(1+11\times i; i=0,1,\dots,9)$, leading to 10 boundaries: 1%, 12%, 23%,

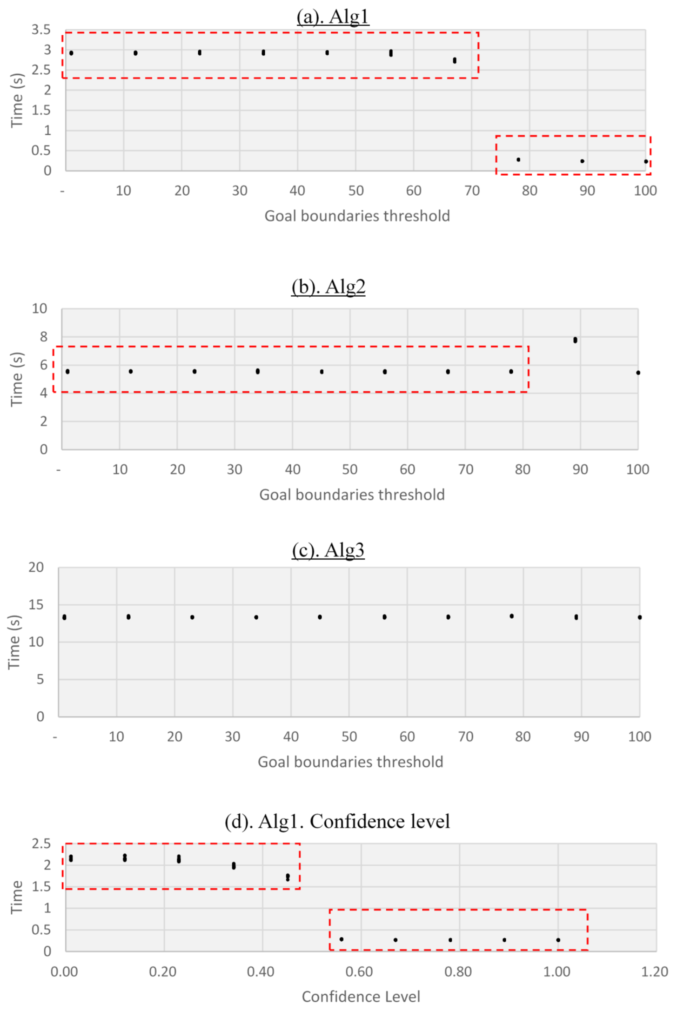


Fig. 6. Runtimes for different criteria's boundaries.

34%, 45%, 56%, 67%, 78%, 89%, 100%. We also investigate the confidence for Algorithm 1, with the same ten percentages. The algorithms were used on the 10 resulting event logs 10 times, and averages are reported here.

Results. As Fig. 6 shows, none of the results suggest a meaningful trend for runtimes when the threshold for the boundaries increases. There are two points in Fig. 6(a) and (b) where the trend of runtime changed. Based on an analysis on the data logs, these phenomena happen in the neighborhood of the average of the random distribution function that generated the synthetic satisfaction levels. Therefore, these phenomena are caused by a data bias.

The same situation and the same bias are shown in Fig. 6(d) for the relation between runtime and the confidence level for Algorithm 1. The figure suggests that out of the neighborhood of the mean of the functions that generated the satisfaction level, there is no meaningful trend for the runtime when the confidence level changes.

H. Evaluation Summary

To answer our research question (Sect. I), the runtime performance of the three GoPED algorithms was assessed in total on 1960 executions over logs with different, controlled characteristics. Six factors were considered to find how the runtime change when the factors change. Table IV summarizes the correlation (positive or negative; linear, quadratic, or none) between the runtimes of our implementation of the algorithms and the six factors.

TABLE IV
CORRELATION BETWEEN ALGORITHM RUNTIMES AND FACTORS

Factor	Algo. 1	Algo. 2	Algo. 3
(L1) Dist. cases / traces	No corr.	No corr.	No corr.
(L2) Number of cases	Pos., Linear	Pos., Quadr.	Pos., Quadr.
(L3) Number of traces	Pos., Quadr.	Neg., Linear	Neg., Linear
(L4) Length of traces	Pos., Linear	Pos., Linear	Pos., Linear
(G1) Num. cons. goals	Pos., Linear	Pos., Linear	Pos., Linear
(G2) Criteria's bounds	No corr.	No corr.	No corr.

The factor that has the most substantial impact on the runtime is the number of cases. This factor is the only factor that has a positive quadratic correlation with the runtime for two Algorithms (2 and 3). Such algorithms select cases by forming and solving an optimization problem. For Algorithm 1, there is a quadratic correlation for the number of traces, but that number is usually much smaller than the number of cases in an event log.

The longest runtime of all 1960 runs was 830 seconds, for the execution of Algorithm 3 on an event log of 500,000 cases, using the aforementioned hardware setup. Such a computer, with a CPU from 2011 and an old operating system, can nowadays be outperformed by a factor of 10 on common laptops. The experiments' runtimes show that the GoPED algorithms are feasible and scalable enough to apply to large event logs, and hence suggest that the algorithms and their implementation can be used in practice.

VI. Related Work

Based on our literature review on goal-oriented mining [11], other reviews on process mining [8], and more recent work published since, we make several observations on work related to our GoPM approach.

GoPM and its GoPED algorithms aim to find a good sequence of activities that satisfy predefined goals, as measured with KPIs. Armentano and Amandi [22] and Yan et al. [23] aim to discover goals from event logs; Santiputri et al. [24] mine goal refinement patterns from multi-layered event logs, and compose these patterns to generate a goal model. Instead, GoPM uses *predefined* goals with relationships as input. Such input goal models can be constructed using conventional RE approaches (e.g., [25], [26]). Intention mining [13] shares some ideas with our work but tries to discover intentional process models, not activity-based process models that satisfy specific goals.

Closer to GoPM, the work of Dąbrowski et al. [14] also suggests combining process mining (on event logs produced by regular and crowdsourced system users) and goals as a means to elicitate requirements. Processes mined from regular user behavior are analyzed for discovering potential goals, which are then used to guide crowd-based process mining experiments for requirements validation. However, this approach does not use goal models as input for case selection in an existing log.

Ponnalagu et al. [27] consider different variants of an industry-scale business process that ideally contribute to achieving the organization's goals. They worked on the admission of a deviating process instance as a valid variant of the intended process that achieves the same goals as the intended process (e.g., a workaround). Although they do not consider a predefined goal model as input, their idea that different variants of a process can be accepted to satisfy a goal is aligned with GoPM's main idea.

Dees et al. [28] proposed a methodology that takes event logs together with a process model (either discovered or designed manually) and *repairs* the process model with respect to the behaviors that do not violate any rule and have a significant improvement on a predefined KPI. This method can be used to align a process model with one goal whose associated KPI is included in the event log. This approach is close to Algorithm 1 in spirit, but limited to one input KPI. Note that there are also repair methods that use goals as input, and in particular Takei and Horita [29] use GoPM to enhance their event log with goal-oriented satisfaction levels before repairing mined Petri Net models.

We also encounter approaches that cluster [30] or filter [31] cases along a specific KPI stored in the event log, enabling discovery, comparisons, and an analysis of deviations from expectations. Such approaches are simplifications of Algorithm 1, as GoPM handles multiple aggregated KPIs (through a goal model) as input, and performs a selection based at the variant level (with a confidence level) instead of the case level.

Note that we are not aware of other techniques that use binary optimization for the selection of variants along aggregated results the way Algorithms 2 and 3 do. The factor-based scalability analysis used here for GoPM's GoPED algorithms is also original.

VII. Limitations and Threats to Validity

This section explores important limitations of GoPM and its GoPED algorithms, as well as threats to the validity of our empirical performance evaluation.

A. Limitations

Absence of goal-related information in event logs. In GoPM, an important new challenge is the absence of real-world logs that include some goal-related attributes (e.g., patient satisfaction in healthcare) besides the usual event characteristics (e.g., timestamps). Although some potential KPIs can be extracted from existing logs (e.g., processing time), the goals of each trace and their satisfaction levels are seldom available and must be inferred from other sources.

Tolerance to noise in event logs. Another challenge relates to the level of precision offered in the three algorithms. Noise, such as meaningless differences in activity sequences, can lead to different traces. For instance, cases from one trace could be selected and those from the other ruled out (all-or-none rule), but the cases from both traces should have been treated the same. The confidence level in Algorithm 1 was meant to mitigate some of the issues related to noise, but its effectiveness in practice remains to be demonstrated on real applications.

Concept drift in the requirements and goals. In GoPED, we assume that the goal models and requirements will not change in unforeseen ways. However, just like processes, goals can also evolve and drift over long durations. At this time, our approach does not handle goals that evolve over time.

B. Threats to Validity

There are also several threats to the validity of our evaluation of the three GoPED algorithms.

Construct validity assesses the degree to which the used evaluation tools are able to answer the research question. One important threat pertains to the use of synthetic event logs to assess the performance and scalability of GoPED. Six different performance-related factors were explored in that experiment. Although these factors cover commonly-used characteristics of event logs, other might exist that would have an impact on the performance as well. In addition, each of these factors was experimented with in isolation, and combinations of two or more factors were not assessed. More performance-related experiments could be performed on existing/benchmark event logs that might also better reflect the complexity of real environments (including noise in the logs). One challenge here would be to augment the logs with realistic goal-oriented information as logs typically do not include explicit goals, KPIs, or goal models.

Internal validity focuses on bias and other confounding factors. One such threat here is that bias might be introduced by having the authors create the event logs, perform the experiments, and analyze the results. We have attempted to be transparent and fair in our experiment setup and analysis as a mitigation; further, our GoPM implementation is openly available for others to perform similar evaluations.

External validity focuses on the extent to which the evaluation results can be generalized to other situations or contexts. We have limited our evaluation to runtime performance, and did not check memory usage along the way. We have also limited our assessment to GRL models (as GRL supports both indicators and numerical satisfaction levels), and our results may not hold for other goal modeling languages. One last but important threat is that the usability of GoPM itself was not assessed by other users.

VIII. Conclusion

This paper presented a Goal-oriented Process Mining approach (GoPM, Fig. 1) that enables the quantitative, goal-driven selection of relevant cases and variants in an event log. We particularly focused on GoPM's three revised GoPED algorithms (for the case, goal, and organization perspectives), which use KPI-informed GRL models as input. An empirical experiment provided positive results related to GoPED's scalability along six factors (Table IV), which suggests practicality on real-sized event logs, despite some documented limitations and threats to the validity of our experiment.

Through our GoPM implementation, the discovery of goal-oriented process models provides a new tool for supporting requirements engineers in better understanding as-is process that meet specific quantitative goals, based on evidence. Future work involves the integration of GoPM to existing process mining tools, as well as usability studies on real event logs and goal models.

References

[1] W. M. P. van der Aalst and J. Carmona, *Process Mining Handbook*, ser. LNBIP. Springer Nature, 2022, vol. 448. [Online]. Available: https://doi.org/10.1007/978-3-031-08848-3

[2] OMG, "Business Process Model and Notation (BPMN), version 2.0.2," 2014. [Online]. Available: https://www.omg.org/spec/BPMN/2.0.2

[3] M. Ghasemi, "What requirements engineering can learn from process mining," in *2018 1st International Workshop on Learning from other Disciplines for Requirements Engineering (D4RE)*. IEEE, 2018, pp. 8–11. [Online]. Available: https://doi.org/10.1109/D4RE.2018.00008

[4] T. M. d. Menezes and A. C. Salgado, "Using logs to reduce the impact of process variability and dependence on practitioners in requirements engineering for traditional business process automation software," *IEEE Access*, vol. 12, pp. 192 874–192 893, 2024. [Online]. Available: https://doi.org/10.1109/ACCESS.2024.3514801

[5] J. Horkoff, F. B. Aydemir, E. Cardoso, T. Li, A. Maté, E. Paja, M. Salnitri, L. Piras, J. Mylopoulos, and P. Giorgini, "Goal-oriented requirements engineering: an extended systematic mapping study," *Requirements engineering*, vol. 24, pp. 133–160, 2019. [Online]. Available: https://doi.org/10.1007/s00766-017-0280-z

[6] D. Amyot, O. Akhigbe, M. Baslyman, S. Ghanavati, M. Ghasemi, J. Hassine, L. Lessard, G. Mussbacher, K. Shen, and E. Yu, "Combining goal modelling with business process modelling: Two decades of experience with the User Requirements Notation standard," *Enterprise Modelling and Information Systems Architectures (EMISAJ)*, vol. 17, pp. 2:1–38, 2022. [Online]. Available: https://doi.org/10.18417/emisa.17.2

[7] K. Diba, K. Batoulis, M. Weidlich, and M. Weske, "Extraction, correlation, and abstraction of event data for process mining," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 10, no. 3, p. e1346, 2020. [Online]. Available: https://doi.org/10.1002/widm.1346

[8] M. Imran, M. A. Ismail, S. Hamid, and M. H. N. M. Nasir, "Complex process modeling in process mining: A systematic review," *IEEE Access*, vol. 10, pp. 101 515–101 536, 2022. [Online]. Available: https://doi.org/10.1109/ACCESS.2022.3208231

[9] M. Ghasemi and D. Amyot, "Goal-oriented process enhancement and discovery," in *International conference on business process management*. Springer, 2019, pp. 102–118. [Online]. Available: https://doi.org/10.1007/978-3-030-26619-6_9

[10] ——, "Data preprocessing for goal-oriented process discovery," in *27th IEEE International Requirements Engineering Conference Workshops (RE 2019)*. IEEE, 2019, pp. 200–206. [Online]. Available: https://doi.org/10.1109/REW.2019.00041

[11] ——, "From event logs to goals: a systematic literature review of goal-oriented process mining," *Requirements Engineering*, vol. 25, no. 1, pp. 67–93, 2020. [Online]. Available: https://doi.org/10.1007/s00766-018-00308-3

[12] T. Takei and H. Horita, "Analysis of business processes with automatic detection of KPI thresholds and process discovery based on trace variants," *Research Briefs on Information and Communication Technology Evolution*, vol. 9, p. 59–76, Sep. 2023. [Online]. Available: https://rebicte.org/index.php/rebicte/article/view/157

[13] G. Khodabandelou, C. Hug, R. Deneckère, and C. Salinesi, "Process mining versus intention mining," in *Enterprise, Business-Process and Information Systems Modeling*. Springer, 2013, pp. 466–480. [Online]. Available: https://doi.org/10.1007/978-3-642-38484-4_33

[14] J. Dąbrowski, F. M. Kifetew, D. Muñante, E. Letier, A. Siena, and A. Susi, "Discovering requirements through goal-driven process mining," in *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*, 2017, pp. 199–203. [Online]. Available: https://doi.org/10.1109/REW.2017.61

[15] ITU-T, "Recommendation Z.151 (10/18) User Requirements Notation (URN) – Language definition," 2018. [Online]. Available: http://www.itu.int/rec/T-REC-Z.151/en

[16] D. Amyot, S. Ghanavati, J. Horkoff, G. Mussbacher, L. Peyton, and E. Yu, "Evaluating goal models within the goal-oriented requirement language," *International Journal of Intelligent Systems*, vol. 25, pp. 841–877, 2010. [Online]. Available: https://doi.org/10.1002/int.v25:8

[17] A. Pourshahid *et al.*, "Business process management with the User Requirements Notation," *Electronic Commerce Research*, vol. 9, pp. 269–316, 2009. [Online]. Available: https://doi.org/10.1007/s10660-009-9039-z

[18] Y. Fan, A. A. Anda, and D. Amyot, "An arithmetic semantics for GRL goal models with function generation," in *International Conference on System Analysis and Modeling*. Springer, 2018, pp. 144–162. [Online]. Available: https://doi.org/10.1007/978-3-030-01042-3_9

[19] R. Conforti *et al.*, "Analysis of business process variants in Apromore," in *BPM Demo Session 2015 [CEUR-WS, Vol. 1418]*. Sun SITE Central Europe, 2015, pp. 16–20. [Online]. Available: https://eprints.qut.edu.au/86669/

[20] M. Ghasemi, "Goal-oriented process mining," Ph.D. dissertation, University of Ottawa, Canada, 2022. [Online]. Available: http://dx.doi.org/10.20381/ruor-27301

[21] N. L. Johnson, S. Kotz, and N. Balakrishnan, *Continuous Univariate Distributions*. Wiley, 1994, vol. 2, ch. Beta distributions, pp. 210–275.

[22] M. G. Armentano and A. A. Amandi, "Towards a goal recognition model for the organizational memory," in *Computational Science and Its Applications – ICCSA 2012*. Springer, 2012, pp. 730–742. [Online]. Available: https://doi.org/10.1007/978-3-642-31137-6_55

[23] J. Yan, D. Hu, S. S. Liao, and H. Wang, "Mining agents' goals in agent-oriented business processes," *ACM Trans. Manage. Inf. Syst.*, vol. 5, no. 4, 2015. [Online]. Available: https://doi.org/10.1145/2629448

[24] M. Santiputri, N. Deb, M. A. Khan, A. Ghose, H. Dam, and N. Chaki, "Mining goal refinement patterns: Distilling know-how from data," in *Conceptual Modeling*. Springer, 2017, pp. 69–76. [Online]. Available: https://doi.org/10.1007/978-3-319-69904-2_6

[25] O. Akhigbe, M. Alhaj, D. Amyot, O. Badreddin, E. Braun, N. Cartwright, G. Richards, and G. Mussbacher, "Creating quantitative goal models: Governmental experience," in *Conceptual Modeling: 33rd International Conference, ER 2014*. Springer, 2014, pp. 466–473. [Online]. Available: https://doi.org/10.1007/978-3-319-12206-9_40

[26] S. Liaskos, R. Jalman, and J. Aranda, "On eliciting contribution measures in goal models," in *2012 20th IEEE International Requirements Engineering Conference (RE)*, 2012, pp. 221–230. [Online]. Available: https://doi.org/10.1109/RE.2012.6345808

[27] K. Ponnalagu, A. Ghose, N. C. Narendra, and H. K. Dam, "Goal-aligned categorization of instance variants in knowledge-intensive processes," in *Business Process Management*. Springer, 2015, pp. 350–364. [Online]. Available: 10.1007/978-3-319-23063-4_24

[28] M. Dees, M. de Leoni, and F. Mannhardt, "Enhancing process models to improve business performance: A methodology and case studies," in *On the Move to Meaningful Internet Systems. OTM 2017 Conferences*. Springer, 2017, pp. 232–251. [Online]. Available: https://doi.org/10.1007/978-3-319-69462-7_15

[29] T. Takei and H. Horita, "Comparison of goal-oriented business process model repair and discovery," *International Journal of Service and Knowledge Management*, vol. 7, no. 1, 2023. [Online]. Available: https://doi.org/10.52731/ijskm.v7.i1.691

[30] G. Sedrakyan, J. De Weerdt, and M. Snoeck, "Process-mining enabled feedback: "Tell me what I did wrong" vs. "tell me how to do it right"," *Computers in Human Behavior*, vol. 57, pp. 352–376, 2016. [Online]. Available: https://doi.org/10.1016/j.chb.2015.12.040

[31] T. Gurgen Erdogan and A. Tarhan, "A goal-driven evaluation method based on process mining for healthcare processes," *Applied Sciences*, vol. 8, no. 6, 2018. [Online]. Available: https://doi.org/10.3390/app8060894