Discovery of Improvement Opportunities in Knock-Out Checks of Business Processes

Katsiaryna Lashkevich(✉), Lino Moises Mediavilla Ponce, Manuel Camargo, Fredrik Milani, and Marlon Dumas

University of Tartu, Narva mnt 18, 51009 Tartu, Estonia
{katsiaryna.lashkevich,linomoises.mediavillaponce,manuel.camargo, fredrik.milani,marlon.dumas}@ut.ee

Abstract. Overprocessing is a source of waste that occurs when unnecessary work is performed in a process. Overprocessing is often found in application-to-approval processes since a rejected application does not add value, and thus, work that leads to the rejection constitutes overprocessing. Analyzing how the knock-out checks are executed can help analysts to identify opportunities to reduce overprocessing waste and time. This paper proposes an interpretable process mining approach for discovering improvement opportunities in the knock-out checks and recommending redesigns to address them. Experiments on synthetic and real-life event logs show that the approach successfully identifies improvement opportunities while attaining a performance comparable to black-box approaches. Moreover, by leveraging interpretable machine learning techniques, our approach provides further insights on knock-out check executions, explaining to analysts the logic behind the suggested redesigns. The approach is implemented as a software tool and its applicability is demonstrated on a real-life process.

Keywords: Business process improvement · Process mining · Overprocessing waste

1 Introduction

Minimizing waste is a common process improvement goal [19]. Overprocessing is a type of waste that occurs when effort is spent performing some activities in the process, but no value is provided to the customer or the business [24]. Overprocessing waste is typical for processes that contain *knock-out checks* [2, 24], i.e., activities that classify cases as either "accepted" or "rejected" and that may then lead to premature termination of a case. When a case is rejected, the work performed on it up to its rejection is the overprocessing waste. Knock-out checks are commonly found in application-to-approval processes in banks (e.g., loan origination processes), insurance companies (e.g., claim-to-settlement processes), and government agencies [2,24].

S. Nurcan et al. (Eds.): RCIS 2023, LNBIP 476, pp. 381–397, 2023.
https://doi.org/10.1007/978-3-031-33080-3_23

Process mining enables data-driven analysis of business processes by analyzing process execution data captured in the form of event logs [1]. By using process mining techniques, analysts can discover the structure of their processes and analyze their performance [13]. In particular, process mining supports the identification and analysis of wastes in business processes [18]. However, few existing process mining-based techniques consider overprocessing waste. In [23], overprocessing is considered, but the approach does not provide transparency as it is based on a black-box model. Thus, existing data-driven techniques for optimizing business processes with knock-out checks either do not consider overprocessing waste or do not provide interpretable results for analysts to consider when seeking to improve the business process.

In this paper, we address this gap by tackling the following research questions (RQ): (1) *How can improvement opportunities related to knock-out checks be identified from event logs?*, and (2) *How can knock-out checks be optimized in a process to reduce the overprocessing waste?* To answer these questions, we propose an interpretable process mining-based approach to (1) discover knock-out checks, their decision rules and dependencies from an event log, (2) identify overprocessing waste associated with knock-out checks, and (3) identify improvement opportunities in knock-out checks and suggest redesigns to reduce overprocessing wastes. The approach has been implemented as a software tool that allows process analysts to upload an event log and obtain overprocessing waste analysis results and redesign suggestions w.r.t. discovered knock-out checks. Our approach is particularly useful for analysts aiming to reduce overprocessing in processes with multiple knock-out checks.

The rest of the paper is structured as follows. Section 2 presents the background and related work. Section 3 describes the proposed approach. Section 4 presents the implementation of the approach. Section 5 covers the evaluation of the approach, and Sect. 6 concludes the paper.

2 Background and Related Work

In this section, we present the concepts of knock-out checks and overprocessing waste. Then, we position our contribution w.r.t. existing approaches to discover, analyze, and redesign knock-out checks.

Knock-Out Checks and Overprocessing. Knock-out checks are activities in business processes that classify cases into two groups: "accepted" and "rejected". When the case is "accepted", it proceeds forward. When "rejected," the case is directed to a designated point of the process known as an anchor. An anchor can be any point in the process, i.e., when the case is rejected, it can be returned to the earlier stage of the process, sent to a later stage, or to process completion [24]. In this paper, we consider knock-out checks that directly (the anchor is the end event of the process) [24] or eventually (the anchor is the activity at a later stage of the process followed by the end event) conclude the case with a negative outcome. A typical example is application-to-approval processes [7], such as a loan application process, where a customer applies for a loan, and the application

goes through several eligibility checks. If a check fails, the application is rejected, and the process execution ends [24].

When a knock-out check results in the rejection of a case, the work performed on the case up to the rejection is considered unnecessary since it delivers value neither to the customer nor to the business [7]. Such unnecessary performed work is a manifestation of an overprocessing waste [22] and, thereby, an inefficiency that results in increased process time and costs [4].

Related Work. A number of studies focus on methodologies for improving business processes [12], in particular, for process redesign. For example, Netjes et al. [14] presented an approach for "Process Improvement by Creating and Evaluating process alternatives" (PriCE), as a tool to support the BPM life-cycle, including the redesign phase. This approach allows analysts to identify applicable redesigns for the selected process parts and to generate alternative process models, alongside their performance evaluation. Likewise, Niedermann and Schwarz [16] introduced a "Deep Business Optimization Platform" that, given a process model and optimization goals, computes recommended changes using a redesign pattern catalog. Souza et al. [20] proposed heuristics to automate the redesign pattern selection. Following the idea of supporting analysts in process redesign, Fehrer et al. [8] proposed a conceptualization of assisted Business Process Redesign (aBPR) and a classification of redesign recommendations by the automation level. These approaches consider collections of redesigns targeting multiple issues, whereas we focus on optimizing knock-out checks to reduce overprocessing. Furthermore, these approaches require an as-is process model as input, whereas we use event logs.

In order to minimize overprocessing waste with knock-out checks, several redesign options have been proposed [11,15,17]. For instance, van der Aalst suggests reordering knock-out checks based on their rejection rate and processing time, combining checks into composite tasks, and placing subsequent checks in parallel [2]. Another heuristic, called the "knock-out principle" [11], proposes ordering knock-out checks according to the "least effort to reject" ratio, i.e., in decreasing order of effort and in increasing order of the rejection rate [17]. Redesign heuristic "early knockout" suggests moving knock-out checks to the earliest possible point of a process [15]. In our approach, we analyze the process to identify when these redesign heuristics can be applied.

Several studies specifically focus on knock-out check optimization. In [2], a set of heuristics for the knock-out process redesign was described, and an implementation of the redesign approach was presented that requires a process model in Petri-Nets notation as input. Verenich et al. [24] proposed a run-time knock-out check reordering technique with predictive models, as opposed to the design-time [2]. Verenich et al. [24] take an event log and information about knock-out checks (the knock-out activity names and the disallowed permutations) as input and suggest how to re-order the knock-out checks on a case-by-case basis. However, this approach is based on black-box predictive models, i.e., does not provide reasoning for the recommended changes.

Process mining is increasingly focusing on the interpretability of the obtained results [10]. Interpretability appears essential for analysts to understand the logic

behind suggested redesigns and predicted performance [6] to gain confidence in making decisions based on recommendations [10]. Thus, Lee [10] proposed an approach for interpretable prediction of process outcomes at run-time from event logs that explains the predicted outcomes. Lashkevich et al. [9] presented an interpretable approach for discovering and analyzing batch processing inefficiencies with insights on batch processing behavior and associated waiting times. In this paper, we also develop an interpretable approach but for the discovery, analysis, and redesign of business processes w.r.t. knock-out checks.

3 Knock-Out Check Discovery and Analysis

In this section, we describe our approach to discover and analyze knock-out check improvement opportunities in a business process. Our approach takes an event log as input to produce a report comprising candidate improvement opportunities relating to knock-out checks and recommended redesigns. Optionally, an analyst can provide insights regarding the process structure as additional input.

Figure 1 depicts an overview of the main steps of the approach. In the first step, we discover knock-out checks, their knock-out rules, and data dependencies from the event log. In the second step, we identify how much overprocessing waste each knock-out check produces and compute their effort-per-rejection rates. In the third and final step, we determine which knock-out checks can be redesigned and, if so, suggestions on how they can be redesigned.

3.1 Knock-Out Check Discovery

The first step of our approach is to discover knock-out checks, their decision rules, and dependencies. As input, we require an event log containing at least case ID, activity label, and one timestamp. With these minimum required data, the approach allows for discovering knock-out checks, identifying improvement opportunities, and presenting redesign options. If the event logs include both start and end timestamps, the approach can also calculate overprocessing waste and, if case attributes are available, knock-out decision rules and dependencies.

Knock-Out Check Discovery. We propose a semi-automated approach for discovering knock-out checks from event logs. First, given an event log, the approach requests analysts to specify either *post-knock-out activities* (one or more

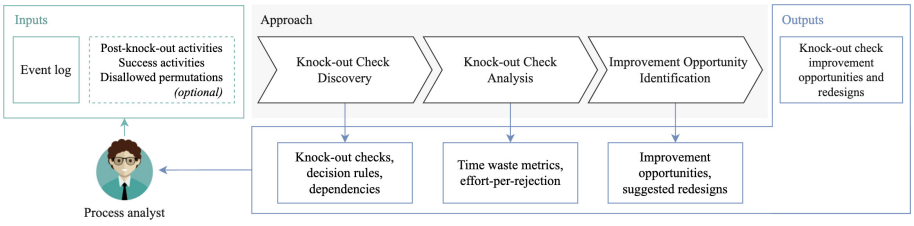

Fig. 1. Overview of the proposed approach.

activities performed immediately after cases are knocked out) or *success activities* (one or more activities performed only for cases that are not knocked out), or both. In addition, analysts can specify *disallowed permutations* (prohibited activity orders). Disallowed permutations, such as activity C cannot be executed before activity B, are useful when dependencies cannot be automatically discovered from the event log (when no relevant case attributes are available). However, manually entered parameters are optional, and analysts can skip this step.

Second, the knock-out check discovery is performed.

(1) If the post-knock-out and/or success activities are specified, the technique identifies knock-out checks as activities preceding post-knock-out and success activities with a directly-follow relation with them.
(2) If the post-knock-out and/or success activities are not specified, the technique extracts the process variants based on the recorded unique pathways and sorts them according to their length. Subsequently, it extracts all transitions between activities in each variant. The transitions are filtered to extract those differentiating for each variant, i.e., those that do not occur in other variants. Among these transitions, the most frequent for each variant is selected and marked as a potential transition that leads to a knockout. These possible transitions are filtered to extract only those activities that led to an early end of the process, i.e., knock-out checks.
(3) If the knock-out checks are known a priori, the analyst can manually mark them. In such cases, tagging post-knock-out and success activities for knock-out discovery is not needed.

As such, the approach supports three modes of operation: (1) semi-automatic discovery with known post-knock-out or success activities, (2) automatic discovery (no input from an analyst is provided, and the discovery is performed exclusively from the event log data), and (3) known knock-out checks (no discovery is performed, and analysts specify the knock-out checks). These operation modes aim to provide flexibility to analysts, allowing them to integrate their domain knowledge of the process.

Decision Rule Discovery. When knock-out checks are identified, we discover the decision rules used to determine which cases are "rejected". For each knock-out check, we obtain a decision rule model and answer the question *"given this case, will the knock-out check reject it?"*, i.e., for every knock-out check, we solve a binary classification problem. We use RIPPER [5], a rule discovery algorithm, to solve the binary classification problem. Other commonly used classification algorithms, such as C4.5 and IREP, were also considered for this task. However, RIPPER has been shown to achieve high accuracy rates in many real-world applications and is generally more efficient than C4.5 and IREP when working with large datasets [5]. This is because RIPPER generates a smaller set of more accurate rules, reducing the computation time needed to classify new instances. Moreover, RIPPER produces a set of rules in natural language with lower complexity (i.e., easier for humans to understand). Since we aim for interpretability in data analysis and decision-making, this algorithm is suitable.

Table 1. Example event log showcasing data dependency.

Case ID	Activity	Start Timestamp	End Timestamp	Amount	Risk Score
1	Check Documents	07/09/2022 16:36	07/09/2022 16:46	35000	—
1	Assess Application	07/09/2022 16:50	07/09/2022 17:30	35000	—
1	Assign Risk Score	07/09/2022 18:10	07/09/2022 18:45	35000	0.56
1	Check Risk Score	07/09/2022 18:45	07/09/2022 18:55	35000	0.56
1	Notify Rejection	07/09/2022 19:00	07/09/2022 19:01	35000	0.56

We create feature vectors by sorting events by end timestamp in ascending order and aggregating event data at the case level, taking the last available value of the attributes of each case. Then, we train the RIPPER decision rule model for every knock-out check and obtain a set of rules in disjunctive normal form, such that if it evaluates to *True* on a given case encoded as a feature vector, the case is labeled as "rejected". For example, consider a knock-out check "Check Liability" and the discovered set of rules `(Monthly Income < 800) V (Owns Vehicle = False)`. If a case has attributes {`Monthly Income: 1200, Owns Vehicle: False, ...`}, the decision rule model of "Check Liability" indicates that this case will be "rejected" in this knock-out check.

Then, the discovered decision rules are filtered based on their confidence; if it is lower than a given threshold, the rules are not taken into consideration for the rest of the analysis. Analysts can also choose to keep them in the analysis with relevant warnings instead. As a result, for each identified knock-out check, we obtain a collection of rules that capture (up to some level of confidence and support) the conditions under which a case is rejected by a knock-out check.

Dependency Discovery. Activities might depend on each other for data and objects. These dependencies must be considered when reordering and relocating activities [2,17,24]. We identify dependencies by using case attributes that appear in the discovered decision rules to perform a search in the log and determine which activity *produces* the value of each case attribute. If the decision rule of a knock-out check K involves a case attribute that is available (or stops changing) after activity A, we consider that the knock-out check K depends on A. For example, consider a loan application process with a knock-out check "Check Risk Score" and knock-out rule `Risk Score > 0.5`. Given a log as in Table 1, we can identify that `Risk Score` is produced by "Assign Risk Score" and "Check Risk Score" depends on "Assign Risk Score" for the `Risk Score` case attribute. Additionally, when the data dependencies cannot be detected from the event log, we allow the user to specify disallowed knock-out check permutations, as in [24].

Thus, the first step results in discovered knock-out checks, their decision rules, and dependencies on other activities based on the case attributes.

3.2 Knock-Out Check Analysis

Once the knock-out checks, their decision rules, and dependencies are discovered, we conduct the knock-out check analysis to answer the questions: (1) *how much*

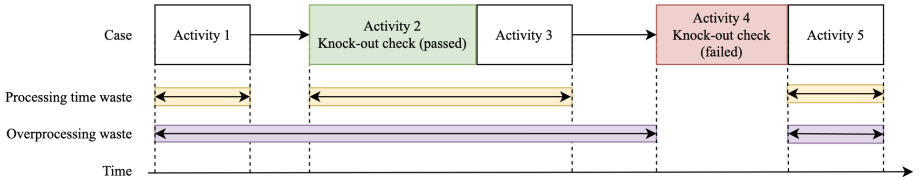

Fig. 2. Processing time waste and overprocessing waste.

waste is associated with cases rejected by each knock-out check? and (2) *what is the mean effort associated with each knock-out check?* This allows us to assess how much overprocessing waste is attributed to each knock-out check and how efficiently they are ordered w.r.t. the "knock-out principle" [11].

When the case is rejected, the effort spent on the case becomes a waste [24]. Therefore, to address question (1), we propose measuring the following metrics: *overprocessing waste*, *processing time waste*, and *waiting time waste*.

Processing time waste is the total effort (processing time) spent on a rejected case, except for the processing time of the check by which it was knocked out. The knock-out check that rejected the case is value-adding since it allows for the termination of an unnecessary case and, thus, is not considered waste [24]. Therefore, the ideal knock-out situation is when an unnecessary case is "rejected" in the first activity (i.e., no overprocessing waste).

Given an application-to-approval process P, a set of resources $R \in P$, a set of activities $A \in P$, a set of knock-out checks $K \subset A$, a set of cases C, a particular case C_i, and a particular knock-out check K_i that rejects C_i,

Definition 1. *The* ***processing time waste*** *for a case C_i due to knock-out check K_i is the sum of the processing times of all activities performed on case C_i, excluding waiting times and the processing time of K_i itself (Fig. 2).*

Overprocessing waste is the total time spent on a rejected case (processing and waiting time), except for the processing time of the check by which it was knocked out. This metric indicates how long the case was in processing before being knocked out.

Definition 2. *The* ***overprocessing waste*** *for a case C_i due to knock-out check K_i as the time elapsed since the case started until it finished, including processing and waiting times but excluding the processing time of K_i itself (Fig. 2).*

Apart from spending effort (processing time) on the rejected cases, the waiting times of "accepted" cases can increase due to the resource being busy processing the cases that are eventually "rejected". Hence, we use the **waiting time waste** metric that defines how much waiting time is induced to the "accepted" cases while the resource processes the "rejected" cases.

Definition 3. *Given the set of cases C_R rejected by K_i and the set of cases C_{NR} not rejected by K_i, we define the* ***waiting time waste*** *associated to K_i*

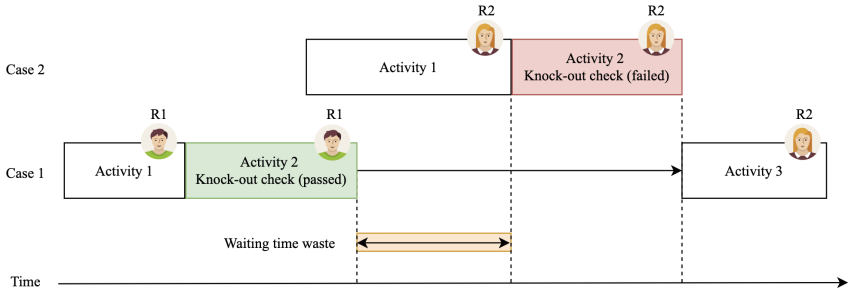

Fig. 3. Waiting time waste. *Case 1* (a non-knocked out case) needs *Resource R2* for advancing, but it has to wait because *Resource R2* is busy on *Case 2* (a case to be knocked out).

as the sum of the duration of the intervals (excluding the processing time of K_i) during which C_{NR} cases are held on standby because the resources responsible for performing activities required to advance them are busy performing work on C_R cases (Fig. 3).

To address question (2), we use the **effort-per-rejection** of each knock-out check that indicates the ratio between the effort spent on the check execution and its rejection probability [11,17]. It describes how much time is spent on the check and how frequently the cases are terminated.

Definition 4. *We define the **effort-per-rejection** of a knock-out check as the ratio between its average processing time and its rejection rate.*

When event logs include only one timestamp, the waste metrics calculation is omitted. In addition, the constant value of processing time is assumed. That is, the effort-per-rejection of knock-out checks in these situations becomes simply the inverse of its rejection rate, similar to what is proposed in [24].

After computing the time waste metrics and effort-per-rejection ratio, we calculate the descriptive statistics for the knock-out checks: (1) *total frequency:* the total number of cases in which the knock-out check is performed, (2) *case frequency:* the proportion of cases in which the knock-out check is performed, (3) *mean duration:* the average duration (including processing and waiting time) of the given knock-out check across all the cases where it has been performed. These statistics provide insights into how frequently the knock-out checks are executed, for how many cases, and how long they take.

3.3 Improvement Opportunity Identification

The final step of our approach is to identify improvement opportunities w.r.t. knock-out checks and suggest possible redesigns. We identify improvement opportunities by examining if there are knock-out check orders, positions of knock-out checks, or different decision rules that can reduce overprocessing. Therefore, we consider the following redesigns:

- **Knock-out reordering**: ordering the knock-out checks by least effort to reject, as in [24].
- **Knock-out relocation**: moving the knock-out checks as early in the process as the case attributes required by their knock-out rules are available (based on the "early knockout" pattern [15]).
- **Knock-out rule adjustment**: changing the value (or range) of numerical attributes of knock-out rules based on the actual distribution of the values observed in the event log.

The knock-out check reordering options are obtained by computing the optimal and dependency-aware ordering of the checks. We do so by applying the *knock-out principle* [11,17], (i.e., in ascending order by their effort-per-rejection value), taking into account any dependencies detected between activities and disallowed permutations if specified by analysts. We, then, use the data dependencies to relocate the knock-out checks in the process, i.e., as early as the case attributes required by their knock-out rules are available. Thus, the technique automatically provides a suggestion on how the knock-out checks should be ordered and relocated to obtain overprocessing reduction.

Finally, for every numerical case attribute appearing in the decision rules of the knock-out checks, we display the distribution of the attribute's values captured in the log and highlight the values of the cases knocked out by a specific check. Although we do not provide suggestions on how the rule should be changed, this data could help analysts in adjusting the knock-out rule values. The result of this step is a report specifying alternative knock-out check orders and positions, and the data on the decision rules that would help to achieve higher temporal efficiency and, in particular, reduce overprocessing waste.

4 Implementation

The proposed approach has been implemented as a software tool and is available on GitHub[1]. Analysts can use this tool to upload event logs with knock-outs and obtain recommendations on how the processes can be redesigned to reduce overprocessing waste. In this section, we illustrate how the approach can be applied on a real-life event log.

For that, we use the environmental permit application process [3]. The log has 1230 cases, 18 activity types, including 3 knock-out checks: "T02-check confirmation of receipt," "T06-determine necessity of stop advice," and "T10-determine necessity to stop indication". These checks have dependencies: "T10" can only be done after either "T02" or "T06" has been performed [24]. Cases in this log contain the following data attributes: the channel by which the case has been lodged, the department that is responsible for the case, the responsible resource, and its group. The log only contains end timestamps.

We uploaded the event log and specified the disallowed permutations (dependencies). Figure 4 depicts the discovered knock-out checks, their total and case

[1] https://github.com/AutomatedProcessImprovement/knockouts-redesign.

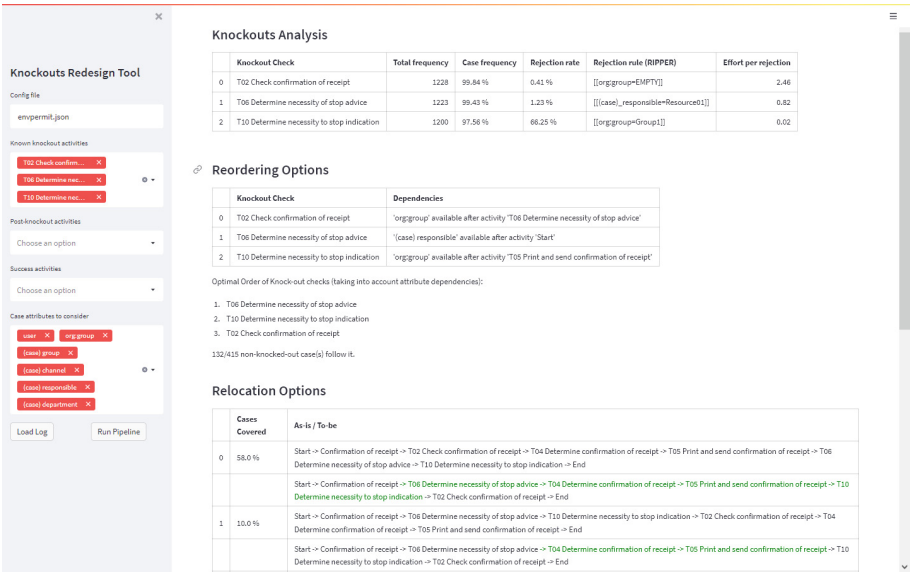

Fig. 4. Screenshot of the tool interface depicting the knock-out check analysis results for the environmental permit application process.

frequencies, rejection rate, decision rule, and effort-per-rejection. The time waste metrics are not calculated since the log has only end timestamps.

Further, the tool provides possible redesign options. The tool computes the optimal ordering of the knock-out checks considering constraints, specifically "T06" -> "T10" -> "T02" (Fig. 4). If "T10" did not require either "T06" or "T02" to be performed before, it would be suggested as the first knock-out check to perform, given its very high rejection rate and consequently low effort-per-rejection value compared to the other checks.

As for the relocation redesign, we observe that the knock-out checks have been placed as early as possible in the process, considering the discovered data dependencies. Figure 4 depicts the relocation option for the most frequent process variants. Theoretically, the knock-out checks could be placed right after the process start since such an order does not violate any discovered dependencies. However, we observed that "Confirmation of receipt" was always performed after the start. Therefore, we marked "Confirmation of receipt" as the start activity of the process, and thus, the relocation did not affect this activity. This showed that in our approach, the dependency detection between activities is limited by the availability and granularity of data.

In this event log, the decision rules of the knock-out checks are based on categorical case attributes (e.g., `org:group=Group1` in Fig. 4). The knock-out rule adjustment redesign focuses on amending numerical case attributes based on which decision rules are formulated (e.g., `Loan Amount > 10000`). Therefore, for this log, no data for the knock-out rule adjustment is presented.

5 Evaluation

In this section, we present the evaluation of our proposed approach, based on the implementation described in the previous section. We evaluate the approach by answering the following evaluation question: (EQ1) *To what extent is the technique able to correctly discover knock-out checks, associated overprocessing waste, and relevant redesigns?* We used synthetic data to address this question and, thereby, validate the ability of the technique to accurately rediscover knock-out checks, overprocessing waste, and improvement opportunities known to be present in the event log (Subsect. 5.1). Further, we compare the accuracy of our approach with the results of the approach proposed in [24], which we consider as baseline (Subsect. 5.2), to answer the following evaluation question: (EQ2) *To what extent the proposed approach sacrifices accuracy (in favor of interpretability) w.r.t. the baseline approach?*

The datasets used in the evaluation and the detailed results are available at https://github.com/AutomatedProcessImprovement/knockouts-redesign.

5.1 Evaluation of Rediscovery Accuracy

To answer EQ1, we created a synthetic event log of a credit application process. The log includes 3000 cases, 6 activity types with resources, start and end timestamps, including 4 knock-out checks: "Check liability", "Check risk", "Check monthly income" and "Assess application" (see Fig. 5). Cases that successfully pass all checks move to "Make credit offer", those that fail any of the checks move to "Notify rejection" and the process is terminated. For all knock-out checks, we assigned rejection rates (see R.R. in Fig. 5) and decision rules. For instance, "Check risk" was assigned a rejection rate of 30% and a decision rule of `(Loan Amount > 10000)`. For that, we post-processed the log to add case attributes with values that reflected the rules and rejection rates.

To test the technique's ability to identify data dependencies, we replicated the situation described in Table 1, namely, "Assess application" can be executed only after "Check risk" is performed. We did so by selectively removing the value

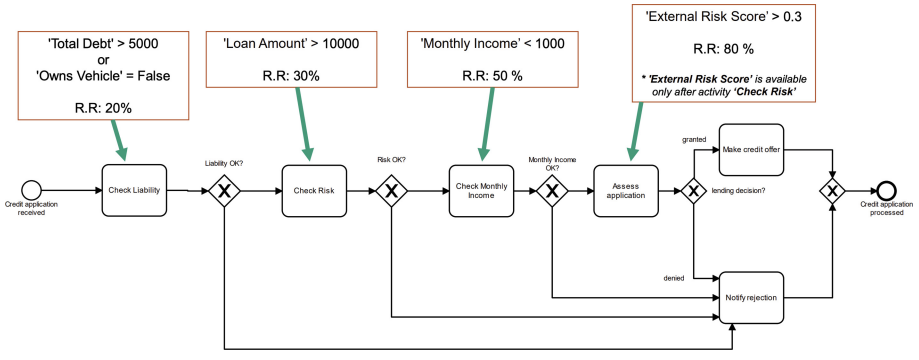

Fig. 5. Synthetic credit application process.

Table 2. Discovered knock-out checks and their decision rules.

Knock-out check	Decision rule	Confidence	Support
Assess application	[[External_Risk_Score=0.36-0.64] V [External_Risk_Score=>0.64]]	1.000	0.732
Check liability	[[Total_Debt=>5219.85] V [Owns_Vehicle=FALSE]]	1.000	0.193
Check monthly income	[[Monthly_Income=<564.21] V [Monthly_Income=564.21-830.79] V [Monthly_Income=830.79-1020.15]]	0.933	0.536
Check risk	[[Loan_Amount=11647.32-16709.71] V [Loan_Amount=>16709.71]]	1.000	0.252

of the "External risk score" attribute such that it becomes available only after "Check risk" is executed. We assess the approach's performance through time-series cross-validation using a temporal split of 80% of the cases for training and 20% for testing on each partition as recommended for time-series data [21].

The technique correctly discovered all four knock-out checks without any input provided by the analyst (the discovery was performed solely from the event log data) and their decision rules (see Table 2). To quantify the performance of the decision rule models obtained for the knock-out checks, we use standard metrics for binary classification performance: the ROC curve and the area under it (AUC). We report the resulting ROC curves and AUC values averaged over five cross-validation folds in Fig. 6. We can observe ROC curves and AUC values near 1.00 (perfect classifier). This observation, together with the high values of confidence (Table 2) and coherence between discovered rules and injected patterns in the log, confirms that the classification models perform as expected.

The technique then calculated the effort-per-rejection rate and the time waste metrics – total overprocessing waste, total processing time waste, and total and mean waiting time waste (see Fig. 7). The correctness of the metrics calculation was verified with a set of unit tests included in the code repository of the approach's implementation.

We verified the correctness of the suggested redesigns by manual comparison against the expected redesigns. The suggested redesigns complied with the expected results. In particular, the technique proposed to re-order the knock-out checks as follows: "Check monthly income" -> "Check risk" -> "Assess

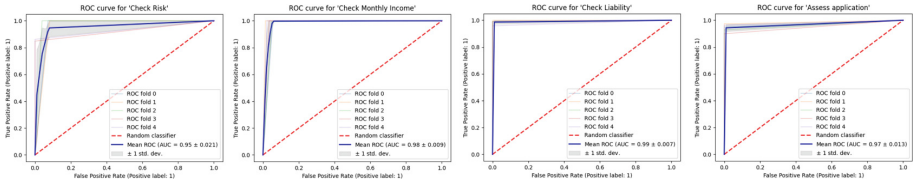

Fig. 6. ROC curves of the decision rule models for the knock-out checks of the synthetic event log.

	Knockout Check	Total frequency	Case frequency	Rejection rate	Rejection rule (IREP)	Effort per rejection	Mean Duration	Total Overprocessing Waste	Total PT Waste	Total Waiting Time Waste	Mean Waiting Time Waste
0	Assess application	836	27.87 %	80.02 %	[[External_Risk_Score=0.35-0.64] V	50.2	1:06:57	178 days, 0:35:20	139 days, 6:45:45	0:00:00	0:00:00
					[External_Risk_Score=>0.64]]						
1	Check Liability	3000	100.0 %	20.17 %	[[Owns_Vehicle=FALSE] V	202.83	1:08:10	55 days, 18:56:48	33 days, 15:40:30	0:00:00	0:00:00
					[Total_Debt=>5200.1]]						
2	Check Monthly Income	1674	55.8 %	50.06 %	[[Monthly_Income=555.77-830.79] V	43.08	0:35:56	139 days, 4:24:45	114 days, 18:13:13	0:00:00	0:00:00
					[Monthly_Income=<555.77] V						
					[Monthly_Income=830.79-1019.68]]						
3	Check Risk	2395	79.83 %	30.1 %	[[Loan_Ammount=11693.71-16840.45] V	136.54	1:08:30	143 days, 1:12:24	93 days, 4:40:17	0:00:00	0:00:00
					[Loan_Ammount=>16840.45]]						

Fig. 7. Knock-out analysis report for the synthetic event log.

application" -> "Check liability". The proposed order indicated that the technique was able to capture the inserted data dependency ("Assess application" depends on the case attribute produced by "Check risk") and consider it for computing the redesigns. If there were no dependencies, "Assess application"' would be ordered before "Check risk" and "Check liability" due to a smaller effort-per-rejection.

All decision rules of the knock-out checks include numerical case attributes (that are expressed as ranges of numerical values). Therefore, for each numerical case attribute of decision rules, the technique provided a graph with the distribution of numerical attribute values and highlighted values of the knocked-out cases. The graphs are coherent with the assigned case attributes and decision rules. For instance, Fig. 8 depicts the graph for the "Total dept" case attribute based on which the knock out in "Check liability" is performed. Using these data, analysts might consider adjusting the knock-out rule by, e.g., rejecting cases with a lower debt amount and, thus, reducing the number of cases processed further in the process and, subsequently, the overprocessing waste. In this way, we have verified that the approach identifies improvement opportunities in the form of redesign recommendations considering data dependencies.

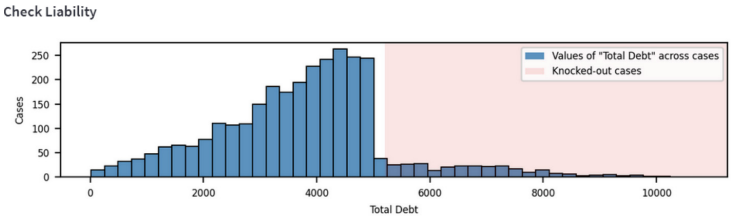

Fig. 8. Data for the knock-out rule adjustment for the synthetics event log.

5.2 Evaluation of Classification Rule Quality

To answer EQ2, we compare our approach with the one proposed by Verenich et al. [24] that we consider the baseline. More specifically, we compare the classification accuracy of the decision rules for knock-out checks discovered by the approach, relative to the classification accuracy of the predictive models used by the baseline [24]. Note that the baseline focuses on run-time optimization using *black-box* models, with an emphasis on classification accuracy. In this research, we focus on design-time optimization and place emphasis on *interpretability*. Accordingly, we expect a priori that the proposed approach would achieve a lower classification accuracy than the baseline, as it sacrifices accuracy in favor of interpretability. What we seek to determine here is the magnitude of the accuracy loss relative to the baseline.

To address this question, we use the same event log as [24], which is an environmental permit application process [3] introduced in Sect. 4. To replicate the experimental conditions of the baseline, we use the same dataset splitting proportions, namely 80% of the cases for training and 20% for testing. Verenich et al. [24] observed that the environmental permit application event log is highly imbalanced regarding the proportions of accepted and rejected cases. Therefore, they performed class balancing by undersampling the accepted (non-knocked-out) cases. We followed this same strategy.

We use the ROC curve and AUC metric to compare the performance of our models with the baseline. To do so, same as Verenich et al. [24], we performed the 5-fold cross-validation procedure using stratified random-sampling-based splitting. We observe AUC values ranging from 0.571 ("T10") to 0.744 ("T02"). These values seem encouraging to a certain extent, especially considering that the AUC values in the baseline ranged from 0.527 ("T06") to 0.645 ("T10") [24]. However, we also observe relatively high standard deviations, e.g., the classifier for "T02" obtained the highest mean AUC value of 0.744 but a standard deviation of 0.37. This can be due to a few positive examples available for training the rule model of this knock-out check: "T02" rejected only 0.4% of all cases, as opposed to "T10", which rejected 64.6%. Nonetheless, the performance of our classification model is comparable to that of the baseline, and no significant performance loss is observed by using an interpretable decision rule-based model instead of black-box models.

This observation suggests that the discovered rules can be used as a proxy for determining how to order knock-out checks in a process (cf. the knock-out reordering strategy in Sect. 3.3) and to relocate the knock-out checks as early as possible in the process (cf. the knock-out relocation strategy).

5.3 Threats to Validity

The observations derived from the above evaluation are subject to the following threats to validity. First, we acknowledge a threat to construct validity due to the

choice of classification accuracy measures (AUC) to address the evaluation questions. While the proposed approach discovers decision rules with relatively high accuracy, comparable to the baseline, this may or may not translate into comparable overprocessing reductions in practical scenarios. Second, we acknowledge a threat to external validity (generalizability) due to the fact that the evaluation relies on one synthetic and one real-life dataset. As such, the conclusions should be seen as preliminary and subject to additional validation.

6 Conclusion

This paper outlines an interpretable process mining approach to identify improvement opportunities in processes containing knock-out checks and recommend redesigns to reduce overprocessing waste. Given an event log, the approach discovers knock-out checks, their decision rules, and dependencies, and calculates time wastes associated with knock-out checks. Next, the approach identifies improvement opportunities and suggests redesigns for reducing overprocessing waste. We present the approach's implementation as a software tool that allows analysts to upload an event log and obtain suggested redesigns. The evaluation shows that our approach can accurately discover and quantify improvement opportunities and suggest relevant redesigns considering data dependencies. Compared to the baseline, we did not lose performance, but we gained explainability. However, the effectiveness of our approach on real-life logs is limited by the availability and granularity of event log data.

In future work, we aim to extend the proposed approach with a simulation phase to uncover further potential improvement opportunities to reduce overprocessing waste. We foresee that a simulation phase would enable us to capture the impact of reordering and relocating knock-out checks for different subsets of cases. Another avenue for future work is to extend the proposed technique to address other types of waste, including waiting waste, transportation waste, and defect waste.

Acknowledgments. This research is funded by the European Research Council (PIX Project).

References

1. van der Aalst, W.M.P.: Process Mining: Data Science in Action, 2nd edn. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49851-4
2. van der Aalst, W.M.: Re-engineering knock-out processes. Decis. Support Syst. **30**(4), 451–468 (2001)
3. Buijs, J.: 3TU. DC dataset: receipt phase of an environmental permit application process (WABO) (2015)
4. Chahal, V., Narwal, M.: Impact of lean strategies on different industrial lean wastes. Int. J. Theor. Appli. Mech. **12**(2), 275–286 (2017)

5. Cohen, W.W.: Fast effective rule induction. In: Machine Learning Proceedings 1995, pp. 115–123. Elsevier (1995)
6. Du, M., Liu, N., Hu, X.: Techniques for interpretable machine learning. Commun. ACM **63**(1), 68–77 (2019)
7. Dumas, M., La Rosa, M., Mendling, J., Reijers, H.A., et al.: Fundamentals of Business Process Management, vol. 1. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-33143-5
8. Fehrer, T., Fischer, D.A., Leemans, S.J., Röglinger, M., Wynn, M.T.: An assisted approach to business process redesign. Decis. Support Syst. **156**, 113749 (2022)
9. Lashkevich, K., Milani, F., Chapela-Campa, D., Dumas, M.: Data-driven analysis of batch processing inefficiencies in business processes. In: Guizzardi, R., Ralyté, J., Franch, X. (eds.) RCIS 2022, pp. 231–247. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-05760-1_14
10. Lee, S.: A rule-based framework for interpretable predictions of business process outcomes using event logs. Master's thesis, Ulsan National Institute of Science and Technology, Ulsan (2021)
11. Lohrmann, M., Reichert, M.: Effective application of process improvement patterns to business processes. Softw. Syst. Model. **15**(2), 353–375 (2016)
12. Malinova, M., Gross, S., Mendling, J.: A study into the contingencies of process improvement methods. Inf. Syst. **104**, 101880 (2022)
13. Milani, F., Lashkevich, K., Maggi, F.M., Di Francescomarino, C.: Process mining: a guide for practitioners. In: Guizzardi, R., Ralyté, J., Franch, X. (eds.) RCIS 2022, pp. 265–282. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-05760-1_16
14. Netjes, M., Reijers, H., Aalst, van der, W.: The price tool kit: tool support for process improvement. In: La Rosa, M. (ed.) Proceedings of the BPM 2010 Demonstration Track, pp. 58–63. CEUR Workshop Proceedings, Springer (2010)
15. Niedermann, F., Radeschutz, S., Mitschang, B.: Design-time process optimization through optimization patterns and process model matching. In: 2010 IEEE 12th Conference on Commerce and Enterprise Computing, pp. 48–55. IEEE (2010)
16. Niedermann, F., Schwarz, H.: Deep business optimization: making business process optimization theory work in practice. In: Halpin, T., et al. (eds.) BPMDS/EMMSAD -2011. LNBIP, vol. 81, pp. 88–102. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21759-3_7
17. Reijers, H.A., Mansar, S.L.: Best practices in business process redesign: an overview and qualitative evaluation of successful redesign heuristics. Omega **33**(4), 283–306 (2005)
18. Reinkemeyer, L.: Process mining in action. Process Mining in Action Principles, Use Cases and Outlook (2020)
19. Rohleder, T.R., Silver, E.A.: A tutorial on business process improvement. J. Oper. Manag. **15**(2), 139–154 (1997)
20. Souza, A., Azevedo, L.G., Santoro, F.M.: Automating the identification of opportunities for business process improvement patterns application. Int. J. Bus. Process. Integr. Manag. **8**(4), 252–272 (2017)
21. Teinemaa, I.: Predictive and prescriptive monitoring of business process outcomes. Doctoral thesis, University of Tartu, Tartu, Estonia (2019)
22. Thürer, M., Tomašević, I., Stevenson, M.: On the meaning of 'waste': review and definition. Prod. Plan. Control **28**(3), 244–255 (2017)

23. Verenich, I.: Explainable Predictive Monitoring of Temporal Measures of Business Processes. Ph.D., Queensland University of Technology (2018). https://doi.org/10.5204/thesis.eprints.124037
24. Verenich, I., Dumas, M., La Rosa, M., Maggi, F.M., Di Francescomarino, C.: Minimizing overprocessing waste in business processes via predictive activity ordering. In: Nurcan, S., Soffer, P., Bajec, M., Eder, J. (eds.) CAiSE 2016. LNCS, vol. 9694, pp. 186–202. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-39696-5_12